CompSci 101
Introduction to Computer Science

March 30, 2017
Prof. Rodger

Announcements

• Assign 6 due, Assign 7 out due April 6
• APT 8 due Tuesday
• APT Quiz 2 Sunday-Tuesday
  – available 6pm or earlier on Sunday
• Exam 2 Tuesday, April 11

• Today:
  – Why are dictionaries so fast?
  – More problem solving with dictionaries
  – Finish problem from last time

Be in the know….
ACM, compsci mailing lists

• Association of Computing Machinery (ACM)
  – Professional organization for computer science
  – Duke Student ACM Chapter – join for free
• Join duke email lists to find out info on jobs, events for compsci students
  – lists.duke.edu – join lists:
    • compsci – info from compsci dept
    • dukeacm – info from student chapter

From Last time - Dictionary

• Consider the Python dictionary below maps schools to number of students in the ACM Club at their school
  d = {'duke':30, 'unc':50, 'ncsu':40, 'wfu':50, 'ecu': 80, 'meridith':30, 'clemson':80, 'gatech':50, 'uva':120, 'vtech':110}

Dictionary to answer which schools have X students? … which schools have groups of students 1-49, 50-99, etc?
Dictionary of **schools** to number students

**Dictionary of number students to schools**

- **ecu**
- **vtech**
- **duke**
- **meridith**
- **wfu**
- **unc**
- **gatech**
- **clemson**
- **ncsu**
- **uva**

- **vtech**
- **duke**
- **meridith**
- **wfu**
- **gatech**
- **unc**
- **ecu**
- **clemson**
- **ncsu**
- **uva**

---

**Inverted Dictionary**

- Start with dictionary of keys to values
  - *Schools to number of students*

- Use it to build an inverted dictionary of values to keys (actually list of keys)
  - *Number of students to list of schools*

- Let's look at the code

---

**Dictionary of number groups to list of schools**

- **0-49**
- **50-99**
- **100-150**

- **duke**
- **Meridith**
- **ncsu**

- **wfu**
- **gatech**
- **unc**

- **ecu**
- **clemson**

- **vtech**
- **uva**

---

**APT EmailsCourse**

You are given a list of strings of course information, where each string is in the format "coursename:person:email". Your task is to determine the course with the most people and to return the emails of those people in the largest course. The emails should be returned as a string with the emails in alphabetical order. If there is more than one largest course, return the emails of such course that comes first in alphabetical order.

```
["CompSci 100: Fred Jack Smith: fjs@duke.edu", 
 "History 117: Fred Jack Smith: fjs@duke.edu", 
 "CompSci 102: Arielle Marie Johnson: amj@duke.edu", 
 "CompSci 100: Arielle Marie Johnson: amj@duke.edu", 
 "CompSci 006: Bertha White: bw@duke.edu", 
 "Econ 051: Bertha White: bw@duke.edu", 
 "English 112: Harry Potter: hp@duke.edu", 
 "CompSci 100: Harry Potter: hp@duke.edu"]
```

Returns "amj@duke.edu fjs@duke.edu hp@duke.edu"
Dictionary Song problem

bit.ly/101s17-0330-3

songs = ["Hey Jude:Let it be:Day Tripper", 
"Let it be:Drive my car:Hey Jude", 
"I want to hold your hand:Day Tripper:Help!", 
"Born to run:Thunder road:She's the one", 
"Hungry heart:The river:Born to run", 
"The river:Thunder road:Drive my car", 
"Angie:Start me up:Ruby Tuesday", 
"Born to run:Angie:Drive my car"]

Assignment 7 – Demo

Snarky, Evil, Frustrating Hangman

- Computer changes secret word every time player guesses to make it "hard" to guess
  - Must be consistent with all previous guesses
  - Idea: the more words there are, harder it is
    - Not always true!

- Example of greedy algorithm
  - Locally optimal decision leads to best solution
  - More words to choose from means more likely to be hung

Canonical Greedy Algorithm

- How do you give change with fewest number of coins?
  - Pay $1.00 for something that costs $0.43
  - Pick the largest coin you need, repeat

Greedy not always optimal

- What if you have no nickels?
  - Give $0.31 in change
  - Algorithms exist for this problem too, not greedy!
Snarky Hangman

• When you guess a letter, you're really guessing a category (secret word "salty") _ _ _ _ _ and user guesses 'a'
  – "gates", "cakes", "false" are all a the same, in 2cd position
  – "flats", "aorta", "straw", "spoon" are all a in different places
• How can we help ensure player always has many words to distinguish between?

Debugging Output and Game Play

• Sometimes we want to see debugging output, and sometimes we don't
  – While using microsoft word, don't want to see the programmer's debugging statements
  – Release code and development code

• You'll approximate release/development using a global variable DEBUG
  – Initialize to False, set to True when debugging
  – Ship with DEBUG = False

Look at howto and categorizing words

• Play a game with a list of possible words
  – Initially this is all words
  – List of possible words changes after each guess

• Given template "_ _ _ ", list of all words, and a letter, choose a secret word
  – Choose all equivalent secret words, not just one
  – Greedy algorithm, choose largest category
Computing the Categories

• Loop over every string in words, each of which is consistent with guess (template)
  – This is important, also letter cannot be in guess
  – Put letter in template according to word
    – _ _ _ a _ t might become _ _ _ a n t

• Build a dictionary of templates with that letter to all words that fit in that template.
• How to create key in dictionary?

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_”

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

See DifferentTimings.py

```python
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 of lists
• Maintain list of [string,count] pairs in order
  
  ```python
  [['ant', 4], ['frog', 2]]
  ```
  – If we read string 'cat', search and update
  
  ```python
  [['ant', 4], ['cat', 1], ['frog', 2]]
  ```
  – If we read string 'dog' twice, search and update
  
  ```python
  [['ant', 4], ['cat', 1], ['dog', 1], ['frog', 2]]
  ```

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

```python
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
```

```python
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 $N$

compsci 101 spring 2017