Administrivia
Course overview
"Document distance" problem
"Peak finding" problem Intro

Handouts
Course info
doc dist python code

Sign up for class at alg.csail.mit.edu
Read collaboration policy!

Course Overview

- Efficient procedures for solving problems on large inputs (e.g., US highway map, human genome)
- Scalability
- Classic data structures and elementary algorithms (CLRS text)
- Real implementations in Python
- Fun problem sets
Content

7 modules, each with motivating problem and problem set (except last)

Linked data structures
Hashing
Sorting
Search
Shortest paths
Dynamic Programming
Numerics

Document Distance Problem

- Given two documents, how similar are they?
  - Identical is easy, plagiarized harder

- Need to define metric
  - Word is sequence of alphanumeric characters

"6.006 is fun" has 4 words

- Word frequencies: \( D(w) = \text{# times } w \text{ occurs in document } D \)
  - Count: \[ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \]
  - \( w \): 6 the is 006 easy fun
\[ D_1 \circ D_2 = \sum_{w} D_1(w) \cdot D_2(w) \] 
inner product
\[ \| D \| = N(D) = \sqrt{D \cdot D} \]
\[ \Theta(D_1, D_2) = \arccos \left( \frac{D_1 \circ D_2}{\| D_1 \| \cdot \| D_2 \|} \right) \]
Identical \( 0 \leq \Theta \leq \pi/2 \) no common words

Python Implementation

docdist1.py

Read file
Make word list
Count frequencies
Sort into order
Compute \( \Theta \)

Jules Verne 25k
Bobsey Twins 268k
Lewis & Clark 1M
Shakespeare 5.5M
Churchill 10M

Expt. Bobsey vs Lewis
\[ \Theta = 0.574 \ (> 2\text{min}) \]

Dies on bigger files

What is going on? Python vs. C? Choice of algorithm?
Profiling

How much time spent in each routine

```
[ import profile
  profile.run("main()")
```

1. # calls
2. tot time: exclusive of subroutine calls
3. per call: \( \frac{2}{1} \)
4. cum: including subroutine calls
5. per call: \( \frac{4}{1} \)

Bobsey vs Lewis

- Total: 1415
- get words from line list: 535
- count-frequency: 505
- get words from string: 125
- Insertion sort: 135

Biggest culprit

```
get_words_from_line_list(L):
    word_list = []
    for line in L:
        words_in_line = get_words_from_string(line)
        word_list = word_list + words_in_line
    return word_list
```

has to be this!
(there isn't anything else here)
**List Concatenation**

\[ L = L_1 + L_2 \]

takes time proportional to \(|L_1| + |L_2|\)

Suppose we had \( n \) lines, each with one word

time proportional to \( 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2} \)

\[ = \Theta(n^2) \]

**Solution:**

\[ \text{word_list} . \text{extend} (\text{words_in_line}) \]

i.e., \( L_1 . \text{extend}(L_2) \)

This takes time proportional to \(|L_2|\)

for each word in \( \text{words_in_line} \)

\[ \text{word_list} . \text{append}(\text{word}) \]

get words from line list: 53.5 s → 0.12 s

**Improvements**

- docdist 1. py
- docdist 2. py
- docdist 3. py
- docdist 4. py
- docdist 5. py
- docdist 6. py
- docdist 6B. py

Original code

<table>
<thead>
<tr>
<th>add profiling</th>
<th>141 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>word_list.extend</td>
<td>94.5 s</td>
</tr>
<tr>
<td>dictionaries in count freq</td>
<td>42.5 s</td>
</tr>
<tr>
<td>process words rather than chars in get_words_from_string</td>
<td>17.5 s</td>
</tr>
<tr>
<td>merge sort rather than insertion sort</td>
<td>6.5 s</td>
</tr>
<tr>
<td>Eliminate sorting by using dictionaries</td>
<td>0.55 s</td>
</tr>
</tbody>
</table>
**PEAK FINDER**

One-dimensional version

\[ \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
a & b & c & d & e & f & g & h
\end{array} \]

a, f, h are numbers

Position 2 is a peak if and only if

\[ b \geq a \quad \text{and} \quad b \geq c \]

Position 8 is a peak if \( h \geq g \)

Problem: Find a peak if it exists