6.006 Fall 2009
L1

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	: Doc distance, peak finding
Hashing	: Selective file update (RSYNC)
Sorting	: gas simulation (?)
Search	: Rubik's cube (?)
Shortest paths	: CalTech → MIT (?)
Dynamic Programming	: Stock Market
Numerics	: $\sqrt{2}$ to infinite precision

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" 4 words
- Word frequencies : $D(w)$ = # times w occurs in document D

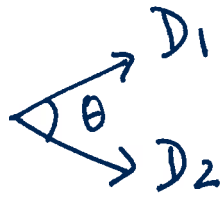
count :	[1	0	1	1	0	1]
w :	6	the	is	006	easy	fun

METRIC

$$D_1 \circ D_2 = \sum_w D_1(w) \cdot D_2(w)$$

inner product

$$\|D\| = N(D) = \sqrt{D \circ D}$$



$$\theta(D_1, D_2) = \arccos\left(\frac{D_1 \circ D_2}{\|D_1\| \cdot \|D_2\|}\right)$$

$$0 \leq \theta \leq \pi/2$$

Identical

no common words

PYTHON IMPLEMENTATION

docdist1.py

Read file

Make word list → ["the", "year", ..]

Count frequencies → [["the", 4012], ["year", 55], ...]

Sort into order [["a", 3120], ["after", 17], ...]

Compute θ

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

Expt: Bobsey vs Lewis

$\theta = 0.574$ (> 2min)

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 : 2/1
4. cum : including subroutine calls
5. per call : 4/1

Bobsey vs Lewis

Total:	141 s
get words from line list :	53 s
count-frequency :	50 s
get words from string :	12 s
insertion sort :	13 s

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 $1 + 2 + 3 + \cdots n = \frac{n(n+1)}{2}$

$= \Theta(n^2)$

Solution: word_list.extend(words_in_line)

i.e., L1.extend(L2)

This takes time proportional to $|L_2|$

for each word in words_in_line word_list.append(word)

get words from line list: 53 s → 0.12 s

IMPROVEMENTS

docdist1.py	original code	141 s
docdist2.py	add profiling	141 s
docdist3.py	word_list.extend	94 s
docdist4.py	dictionaries in count freq	42 s
docdist5.py	process words rather than chars in get words from string	17 s
docdist6.py	merge sort rather than insertion sort	6 s
docdist6B.py	Eliminate sorting by using dictionaries	0.5 s

PEAK FINDER

One-dimensional version

1	2	3	4	5	6	7	8
a	b	c	d	e	f	g	h

a to h are numbers

Position 2 is a peak if and only if $b \geq a$ and $b \geq c$

Position 8 is a peak if $h \geq g$

Problem: Find <u>a</u> peak if it exists