6.006
Spring 2008
L1

Administrivia

Course Overview

"Document distance" problem

Handouts

#1 Course-info
#2 Signup sheet
#3 Doc distance

READ COLLABORATION POLICY!

PRE-REQS: Python, discrete mathematics

Will have 4 times that minimize conflicts

Course Overview

- Efficient procedures for solving problems on large inputs (entire works of Shakespeare, human genome, US highway map)
- Scalability
- Classic data structures and elementary algos (CLRS text)
- Real implementations in Python/ Fun problem sets
- β version of class — give us feeback!

Content

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

Linked data structures	:	Document distance (DD)
Hashing	:	DD, genome comparison
Sorting	:	gas simulation
Search	:	Rubik's cube 2x2x2
Shortest paths	:	Caltech → MIT
Dynamic Programming	:	Stock market
Numerics	:	$\sqrt{2}$

Document Distance Problem

- Given two documents, how similar are they?
  - identical - easy?
  - modified or related (DNA, plagiarism, authorship)
  - Did Francis Bacon write Shakespeare's plays?
- 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

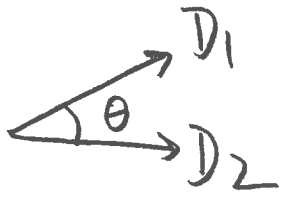
choose some canonical ordering of words

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

---

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 10M.

$\theta = 0.574$ (3 minutes)

Expt: Bobsey vs Lewis
Dies on bigger files

What is going on?

Python vs C?
Choice of algorithm
$\theta(n^2)$ versus $\theta(n)$?

Profiling

How much time spent in each routine

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

① #calls
② tottime: exclusive of subroutine calls
③ per call: ②/①
④ cum: including subroutine calls
⑤ percall: ④/①

Bobsey vs Lewis

195 secs total.
107: get words from line list
44: count-frequency
13: get words from string
12: insertionsort

```
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 $|L_1| + |L_2|$

if 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)

[word_list.append(word)] for each word in words_in_line

L1.extend(L2) time proportional to $|L_2|$

Python has powerful primitives built in. To write efficient algorithms, we need to understand their costs. Figuring out cost of set operations in PS 1 ...

---

docdist3.py

195 secs → 85 secs.

```
def count_frequency(word_list)
    L = []
    for new_word in word_list:
        for entry in L:
            if entry[0] == new_word:
                entry[1] = entry[1] + 1
                break
        else:
            L.append([new_word, 1])
    return L
```

Analysis:

n words
d distinct words
$\Theta(nd)$
if all words distinct $\Theta(n^2)$

Dictionaries

Hash tables: mapping from domain (finite collection of immutable things) to range (anything). D['ab'] = 2 D['the'] = 3

docdist4.py uses dictionaries that give constant time lookup. 85 secs → 42 secs

see count_frequency. time $\Theta(n)$

```
D = {}
for new_word in word_list:
    if D.has_key(new_word):
        D[new_word] = D[new_word]+1
    else:
        D[new_word] = 1
return D.items()
```

What's left? get_words_from_string 13 secs
(V5 fixes with translate)

insertion_sort 11 secs
(V6 fixes with merge_sort)

next time