(turn cellphone off!)

6.006
Rivest
L1.1
9/4/08

Outline:

- ☐ Administrivia
- ☐ Course overview
- ☐ "Document distance" problem

Handouts:
1. Course info
2. sign-up sheet (passed around)
3. docdist1.py

Administrivia:

- Welcome to 6.006!
- Introduce staff
- sign-up – on line
  and – on sheet being passed around
- web = http://courses.csail.mit.edu/6.006
- Ask students to raise hands for:
  - credit/listener/unsure
  - frosh/soph/...
  - MIT/not
  - course 6 / 18 other
  - python
  - 6.01
  - 6.042
- Pre-reqs: see TA's if haven't got them, but have ≡
- Recitations: on-line; get assignment tonight – bring laptops!
- lectures/recitations/problem sets/2 quizzes/final
- textbook: CLRS, rec: Miller/Ranum
- relation to 6.046

Course Overview

- Efficient procedures for solving problems on large inputs
- scalability (now can have, & we'll use complete works of Shakespeare, human DNA, or U.S. highway map on your laptop)
- classic data structures & elementary algorithms (CLRS)
- real implementations (Python)
- having some fun (problem sets!)
- developing this course:
  - this is only [illegible] version – will have rough edges
  - we want your feedback – think of yourselves as "co-designers"

Content:

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

• Intro & linked data structures:	Document Distance	Set Opns
• Hashing	"	
• Dynamic Programming	Image Resizing →	Chimp DNA
• Sorting	Heapsort	Gas Simulation
• Search	BFS/DFS	2x2x2 Rubik
• Shortest Paths		CalTech→MIT
• Numerics	Least squares	

6.006
Rivest
L1.3
9/4/08

Document Distance Problem (Document Similarity)

- Given two "documents" how similar are they? common problem
  - identical - easy?
  - modified or related (DNA, plagiarism, authorship)
  - Did Francis Bacon write Shakespeare's plays?
  - Need to define metric
- define word = sequence of alphanumerics "6.006 is fun" 4 words
- word frequencies: $D(w) =$ # times w occurs in document D

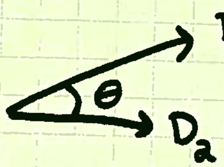
Can think of frequencies as a vector, each possible word is a coordinate

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

for some canonical ordering of of words...

$$D_1 \cdot D_2 = \sum_{w} D_1(w) \cdot D_2(w) \qquad \text{inner product}$$

$$\|D\| = N(D) = \sqrt{D \cdot D} \qquad \text{length, norm}$$



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

$$0 \le \theta \le \pi/2$$

↑ identical ↑ no common words

Problem: given D_1, D_2 compute θ (angle between their word frequency vectors)

6.006
L1.4
Rivest
9/4/08

		(bytes)
• Data Sets:	Jules Verne "2889"	25K
	Bobsey Twins	268K
	Lewis & Clark	1M
	Arabian Nights	3M
	Churchill	10M
	Shakespeare	5.5M
	Bacon	320K

Project Gutenberg

(You can also try reading these!)
(esp. Verne) → read excerpt

Procedure (docdist1):

x2:
- read file
- make word list ["the", "year", "2889", "by", "Jules", ...]
- count frequencies [["the", 4012], ["year", 55], ...]
- sort into order [["a", 3120], ["after", 17], ...

- compute angle = $\arccos\left(\dfrac{D_1 \cdot D_2}{\|D_1\| \cdot \|D_2\|}\right)$

Look at Python code:

- students should be able to read this code (ask TA if not, in recit.)
- go through routines (admittedly not most efficient, but correct.)

Experiment: Bobsey vs Lewis $\Theta = 0.574$ (3 minutes)
Doesn't seem to scale well, just "dies" on bigger files... (too long)

What is going on?

Discuss:
- Python vs C? (choice of programming lang.) x10 or so only
- choice of algorithm $\Theta(n^2) \Rightarrow \Theta(n)$ 10^3 or more speedup
  much more important

6.006
L1.5
Rivest
[redacted]
9/4/08

Profiling:

- How much time is spent in each routine :
  ~~total exclusive of subroutine time~~
  ~~percall~~

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

① # calls
② tottime - exclusive of subroutine calls
③ percall - ②/①
④ cum - including subroutine calls
⑤ percall - ④/①

Results: ~~B~~ Bobsey vs Lewis

- 194 seconds total - 3 minutes!
- 107 in get words from line list
- 44 in count_frequency
- 13 in get words from string
- 12 in insertion-sort

exclusive of subroutine calls they make

eventually we'll get all this down to 6 seconds

```
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!)

!?

6.006
Rivest
L1.6
9/4/08

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 $1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2} = \Theta(n^2)$

Solution:

word_list.extend(words_in_line) ← time proportional to len(words_in_line)

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

Python has powerful primitives built-in.

To write efficient algorithms, we need to understand their costs.

(Figuring out cost of set opns will be their homework...)

6.006
Rivest
[redacted] L1.7
9/4/08

Document Distance

t2.bobsey.txt 263 KB
t3.lewis.txt 1 MB

version 1:	initial	? secs ✓
2:	add profiling	194 secs ✓
3:	wordlist.extend(words_in_line)	84 secs ✓
4:	use dictionaries in count_frequency	41 secs
5:	translate	13 secs
6:	merge_sort	6 secs

with docdist3:

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

[["the", 2509], ["on", 369], ...

try:
```
from docdist3 import *
L = read_file("t2.bobsey.txt")
W = get_words_from_line_list(L)
F = count_frequency(W)
```

L[0], L[-7)
W[:7], W[-7:7]
(slow on lewis - don't try in class)

Analysis: time $= O(n \cdot d)$

n words
d distinct words

if all words distinct, $d = n$

time $= \Theta(n^2)$

6.006
Rivest
[redacted] L1.8
[redacted]
9/4/08

Dictionaries – Hash Tables

mapping from domain (finite collection of immutable things)
to range (anything)

```
D = { }                  empty mapping
D['ab'] = 2
D['the'] = 3             fast!
D
D['ab']                  fast!
D['xyz']                 error
D.has_key('xyz')
D.items ()
D.keys ()
```

show docdist4 count_frequency

cuts time in 1/2 (more for larger files) (84 secs ⇒ 41 secs)

Analysis: time $\Theta(n)$ doesn't depend on what's already in table

Remaining time goes to:

get_words_from_string (v5 fixes with translate) 13 secs

insertion_sort (v6 fixes with merge-sort) 11 secs

Important to understand costs (running times) of Python primitives!

See <u>Python Cost Model</u> (web site) for some experimentation...

(You'll mimic this for Python <u>set</u> operations...) HW #1