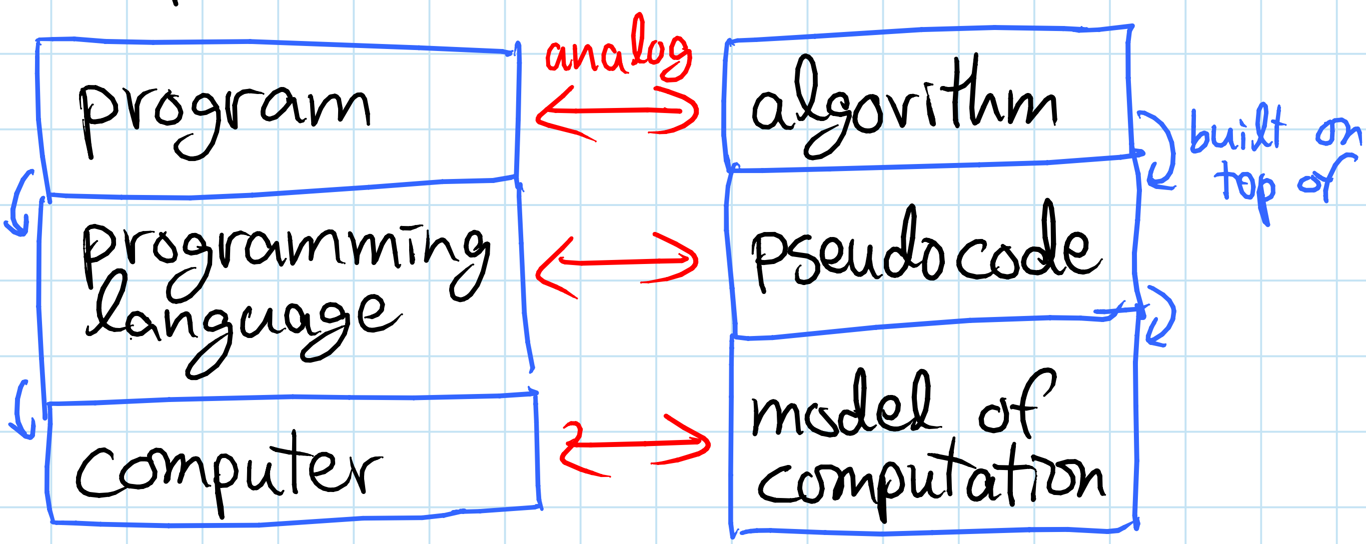
6.006	Lecture 2	Sept. 13. 2011

TODAY: Models of Computation
- what's an algorithm? what is time?
- random access machine
- pointer machine
- Python model
- document distance: problem & algorithms

History: al-Khwārizmī "al-kha-raz-mi" (c. 780-850)
- "father of algebra" with his book "The Compendious Book on Calculation by Completion & Balancing"
- linear & quadratic equation solving: some of the first algorithms

http://en.wikipedia.org/wiki/Al-Khwarizmi

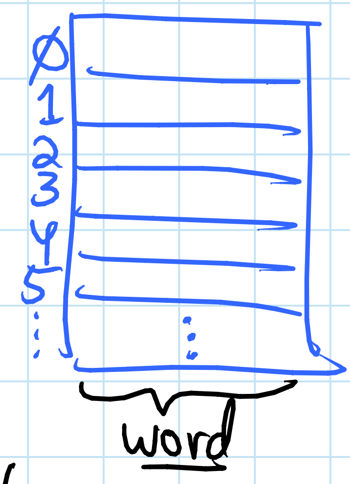
What's an algorithm?
- mathematical abstraction of computer program
- computational procedure to solve a problem



Model of computation specifies
- what operations an algorithm is allowed
- cost (time, space, ...) of each operation

→ cost of algorithm = sum of op. costs

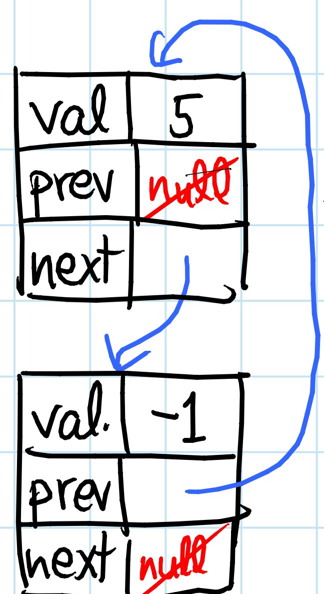
① Random Access Machine (RAM):



- Random Access Memory (RAM) modeled by a big array
- $\Theta(1)$ registers (each 1 word)
- in $\Theta(1)$ time, can
  - load word @ r_i into register r_j
  - compute (+, −, *, /, &, |, ^) on registers
  - store register r_j into memory @ r_i
- what's a word? $w \geq \lg(\text{mem. size})$ bits
  - assume basic objects (e.g. int) fit in word
  - Unit 4 deals with big numbers
- realistic & powerful ⇝ implement abstractions

② Pointer Machine:

(namedtuple)



- dynamically allocated objects
- object has $O(1)$ fields
- field = word (e.g. int) or pointer to object/null (a.k.a. reference)
- weaker than (can be implemented on) RAM

Python lets you use either mode of thinking:

1. "list" is actually an array → RAM
   - $L[i] = L[j] + 5$ → $\Theta(1)$ time
2. object with $O(1)$ attributes → pointer machine
   ↳ including references
   - $x = x.next$ → $\Theta(1)$ time

Other operations: Python has many
- to determine their cost, imagine implementation in terms of 1 or 2

list:
- L.append(x) → $\Theta(1)$ time
  - obvious if you think of infinite array
  - but how would you have >1 on RAM?
  - via table doubling [Lecture 9]

- L=L1+L2 ≡
```
L = []                 } Θ(1)
for x in L1:
    L.append(x)        } Θ(1)  } Θ(|L1|)
for x in L2:
    L.append(x)        } Θ(1)  } Θ(|L2|)
```
  ↓
  $\Theta(1 + |L_1| + |L_2|)$ time

- L1.extend(L2) ≡
```
for x in L2:
    L1.append(x)       } Θ(1)  } Θ(1+|L2|) time
```
  ≡ L1 += L2

- L2 = L1[i:j] ≡ L2 = []
```
for k in range(i, j):
    L2.append(L1[i])
```
$\}\Theta(1)$ $\}\Theta(j-i+1) = O(|L|)$

- b = x in L ≡
```
for y in L:
    if x == y:
        b = True
        break
else:
    b = False
```
& L.index(x)
& L.find(x)

$\}\Theta(1)$ $\}\Theta(\text{index of } x) = O(|L|) = \Theta(|L|)$ in worst case

- len(L) → $\Theta(1)$ time
  - list stores its length in a field

- L.sort() → $\Theta(|L| \lg |L|)$
  - via comparison sort [Lecture 3 (& 4 & 7)]

tuple, str: similar (think of as immutable lists)

dict: D[key] = val, key in D $\}$ $\Theta(1)$ time with high probability
  - via hashing [Unit 3 = Lectures 8–10]

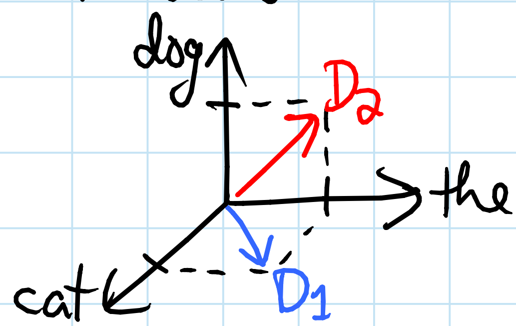
set: similar (think of as dict without vals.)

heapq: heappush & heappop → $\Theta(\lg n)$ time
  - via heaps [Lecture 4]

long: x+y → $O(|x|+|y|)$ time (#words)
x*y → $O((|x|+|y|)^{\lg 3})$ time ($\lg 3 \approx 1.58$)
  - via Karatsuba algorithm [Lecture 11]

Document distance problem: compute $d(D_1, D_2)$

- applications: find similar documents
  detect duplicates & plagiarism
  web search (D_2 = query)
  Wikipedia mirrors & Google
- word = sequence of alphanumeric chars.
- document = sequence of words
  (ignore space, punctuation, etc.)
- idea: define distance in terms of shared words
- think of document D as vector:
  $D[w]$ = # occurrences of word w
- e.g.: D_1 = "the cat"
  D_2 = "the dog"



- attempt 1: $d'(D_1, D_2) = D_1 \cdot D_2 = \sum_w D_1[w] \cdot D_2[w]$
  - problem: not scale invariant
    $\Rightarrow$ long docs. with 99% same words seem farther than short docs. with 10%
- fix: normalize by # words: $d''(D_1, D_2) = \frac{D_1 \cdot D_2}{|D_1| \cdot |D_2|}$ ← # words
- geometric (rescaling): $d(D_1, D_2) = \arccos d''(D_1, D_2)$
  = angle between vectors
  0 = same
  90° = not

[Salton, Wong, Yang 1975]

Document distance algorithm:

1. split each document into words
2. count word frequences (document vectors)
3. compute dot product (& divide)

①: re.findall(r"\w+", doc) → what cost?

~ in general, re can be exponential time!

```
→ for char in doc:                              } Θ(|doc|)
      if not alphanumeric:
          add previous word
            (if any) to list                    } Θ(1)
          start new word
```

```
②: sort word list        ← O(k lg k · |word|)
                              ↳ #words   ↑ to compare
   for word in list:                                   } O(Σ |word|) = O(|doc|)
       if same as last word: ← O(|word|)
           increment counter                           } Θ(1)
       else:
           add last word & count to list
           reset counter to Ø
```

```
③: for word, count1 in doc1:       ← Θ(k_1)      } O(k_1 · k_2)
       if word, count2 in doc2:    ← O(k_2)
           total += count1 * count2  } O(1)
```

③': start at first word of each list
if words equal: $\leftarrow O(|word|)$
    total += count1 * count2
if word1 ≤ word2: $\leftarrow O(|word|)$
    advance list1
else:
    advance list2
repeat until either list done

$O(\Sigma|word|) = O(|doc|)$

Dictionary approach:

②': count = {}
for word in doc:
    if word in count: $\leftarrow \Theta(|word|) + \Theta(1)$ w.h.p.
        count[word] += 1
    else:
        count[word] = 1

$\Theta(1)$

$O(|doc|)$ with high prob.

③ as above $\rightarrow O(|doc_1|)$ w.h.p.

Code: (lecture2_code.zip & _data.zip on website)

t2.bobsey.txt 268,778 chars/49,785 words/3,354 uniq
t3.lewis.txt 1,031,470 chars/182,355 words/8,530 uniq

seconds on Pentium 4, 2.8GHz, C-Python 2.6.2, Linux 2.6.26

- docdist1: 228.1 - 1, 2, 3 (with extra sorting)
  - words = words + words_on_line
- docdist2: 164.7 - words += words_on_line
- docdist3: 123.1 - 3'... with insertion sort
- docdist4: 71.7 - 2' but still sort to use 3'
- docdist5: 18.3 - split words via string.translate
- docdist6: 11.5 - merge sort (vs. insertion)
- docdist7: 1.8 - 3 (full dictionary)
- docdist8: 0.2 - whole doc., not line by line