Lecture 2: Models of Computation

Lecture Overview

- What is an algorithm?
- Random access machine
- Pointer machine
- Python model
- Document distance: problem & algorithms

History

Al-Khwārizmi “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

What is 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 operation costs

Random Access Machine (RAM)

• Random Access Memory (also abbreviated RAM) modeled by a big array
• Θ(1) registers (each 1 word)
• In Θ(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? \( w \geq \log \) (memory size) bits to store address
  – assume basic objects (e.g., int) fit in word
  – Lectures 11 and 12 in the course deal with big numbers
• realistic and powerful → implement abstractions
Pointer Machine

- 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 Model

Python lets you use either mode of thinking

1. “list” is actually an array $\rightarrow$ RAM
   
   \[ L[i] = L[j] + 5 \rightarrow \Theta(1) \text{ time} \]

2. object with $O(1)$ attributes (including references) $\rightarrow$ pointer machine
   
   \[ x = x.next \rightarrow \Theta(1) \text{ time} \]

Python has many other operations. To determine their cost, imagine implementation in terms of (1) or (2):
1. list
   (a) \( L.append(x) \rightarrow \theta(1) \) time
       obvious if you think of infinite array
       but how would you have \( > 1 \) on RAM?
       via table doubling [Lecture 9]

   (b) \[ L = L_1 + L_2 \] \( \equiv \) \( L = [\] \rightarrow \theta(1) \)
       \( \text{for } x \text{ in } L_1: \)
       \( \text{L.append(x) } \rightarrow \theta(1) \) \( \{ \theta(|L_1|) \} \)
       \( \text{for } x \text{ in } L_2: \)
       \( \text{L.append(x) } \rightarrow \theta(1) \) \( \{ \theta(|L_2|) \} \)

   (c) \( L_1.extend(L_2) \equiv \text{for } x \text{ in } L_2: \)
       \( \text{L1.append(x) } \rightarrow \theta(1) \) \( \{ \theta(1 + |L_2|) \text{ time} \} \)

   (d) \( L_2 = L_1[i : j] \equiv L_2 = [\] \)
       \( \text{for } k \text{ in range}(i, j): \)
       \( \text{L2.append(L1[i]) } \rightarrow \theta(1) \) \( \{ \theta(j - i + 1) = O(|L|) \} \)

   (e) \( b = x \text{ in } L \equiv \text{for } y \text{ in } L: \)
       \( \& \text{L.index(x)} \text{ if } x == y: \)
       \( \& \text{L.find(x)} \)
       \( b = True; \)
       \( \text{break} \)
       \( \text{else} \)
       \( b = False \)
       \( \theta(\text{index of } x) = \theta(|L|) \)

   (f) \( \text{len(L)} \rightarrow \theta(1) \) time - list stores its length in a field

   (g) \( \text{L.sort()} \rightarrow \theta(|L| \log |L|) \) - via comparison sort [Lecture 3, 4 & 7]

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

3. dict: via hashing [Unit 3 = Lectures 8-10]
   \( D[key] = \text{val} \)
   \( \text{key in } D \) \( \{ \theta(1) \text{ time w.h.p.} \} \)

4. set: similar (think of as dict without vals)
Document Distance Problem — compute $d(D_1, D_2)$

The document distance problem has applications in finding similar documents, detecting duplicates (Wikipedia mirrors and Google) and plagiarism, and also in web search ($D_2 = \text{query}$).

Some Definitions:

- **Word** = sequence of alphanumeric characters
- **Document** = sequence of words (ignore space, punctuation, etc.)

The idea is to define distance in terms of shared words. Think of document $D$ as a vector: $D[w] = \#$ occurrences of word $w$. For example:

![Diagram](image_url)

Figure 2: $D_1 = \text{“the cat”}$, $D_2 = \text{“the dog”}$
As a first attempt, define document distance as

\[ d'(D_1, D_2) = D_1 \cdot D_2 = \sum_w D_1[w] \cdot D_2[w] \]

The problem is that this is not scale invariant. This means that long documents with 99\% same words seem farther than short documents with 10\% same words. This can be fixed by normalizing by the number of words:

\[ d''(D_1, D_2) = \frac{D_1 \cdot D_2}{|D_1| \cdot |D_2|} \]

where \(|D_i|\) is the number of words in document \(i\). The geometric (rescaling) interpretation of this would be that:

\[ d(D_1, D_2) = \arccos(d''(D_1, D_2)) \]

or the document distance is the angle between the vectors. An angle of 0° means the two documents are identical whereas an angle of 90° means there are no common words. This approach was introduced by [Salton, Wong, Yang 1975].

**Document Distance Algorithm**

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

Define:
\[
\begin{align*}
  n & : \#\text{words} \\
  |word| & : \#\text{characters in word} \\
  |doc| & : \#\text{characters in document} = \sum |word| \\
\end{align*}
\]

(1) Splitting:
\[
\begin{array}{ll}
  \text{for char in doc:} & \Theta(|doc|) \\
  \quad \text{if not alphanumeric} & \Theta(1) \\
  \quad \text{add previous word} & \Theta(1) \\
  \quad \text{(if any) to list} & \Theta(1) \\
  \quad \text{start new word} & \Theta(1) \\
\end{array}
\]
(2) sort word list \( \leftarrow O(n \log n \cdot |word|) \) where \( n \) is \#words

for word in list:
    if same as last word:
        increment counter
    else:
        add last word and count to list

\[ O(\sum |word|) = O(|doc|) \]

\( \Theta(1) \)

if words equal:
    \( \Theta(1) \)

\( \Theta(n) \) \( \Theta(n_1 \cdot n_2) \)

(3) for word, count1 in doc1:
    \( \Theta(n_1) \)

if word, count2 in doc2:
    \( \Theta(n_2) \)

\[ \text{total} += \text{count1} \cdot \text{count2} \]

\( \Theta(1) \)

\( O(\sum |word|) = O(|doc|) \)

Dictionary Approach

(2)' count = {} 

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

\( \Theta(1) \)

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