Alin Tomescu | Week 1, Thursday, February 6th, 2014 | Lecture 2
6.006 Intro to Algorithms | Prof. Srinivas Devadas | Prof. Nancy Lynch | Prof. Vinod Vaikuntanathan

Lecture 2: Document distance

Lecture overview

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

What is an algorithm

The term "**algorithm**" is derived from the Latin translation of Al Khwarizmi's name.

You can think of an algorithm as a mathematical abstraction of a computer program. You can think of it as a computational procedure to solve a problem.

Program	↔	*Algorithm*
Programming language	↔	Pseudocode
Computer	↔	Model of computation

What **operations** is an algorithm allowed? What is the cost of each of these operations (time, space)? Mostly we care about time complexity.

The **cost** of an algorithm is defined as the sum of its operation's costs.

Random access machine (RAM)

A **random access machine** assumes there is random access memory in it (a big array).

Register address	**Register contents**
0	$word1$
1	$word2$
2	…
3	
$2^k - 1$	

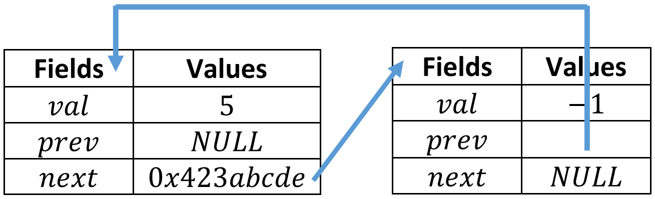
In RAM memory you are able to:

- Access or load a word into a register in $\Theta(1)$ time, does not matter which register.
- Compute certain things on registers ($+, -, /$, etc.), and the operations all take $\Theta(1)$ time.

The word size $w \geq \log_2(\text{max. memory size})$ usually.

The pointer machine

Dynamically allocated objects. Object has $\Theta(1)$ fields



Fields	**Values**
val	5
$prev$	$NULL$
$next$	$0x423abcde$

Fields	**Values**
val	-1
$prev$	
$next$	$NULL$

This model is *weaker* in the sense that it can be implemented in the RAM model.

Python model

List model

Lists in Python are actually arrays.

- $L[i] = L[j] + 5$ takes $\Theta(1)$ time

Pointer model

You have an object with a number of *attributes* or *fields*. Accessing a field takes $\Theta(1)$ time.

- $x = x.next$ takes $\Theta(1)$ time

Python lists

(a) Appending an element to a list: $L1.append(x)$ **takes $\boldsymbol{\Theta(1)}$ time**

(b) Adding lists together: $L = L1 + L2$
   - a. An empty list L is created
   - b. Every element in $L1$ is appended to L – $|L1| \cdot \Theta(1) = \Theta(|L1|)$
   - c. Every element in $L2$ is appended to L - $|L2| \cdot \Theta(1) = \Theta(|L2|)$
   - d. Whole thing **takes $\boldsymbol{\Theta(|L1| + |L2|)}$**

(c) $L1.extend(L2)$ adds all elements of $L2$ to $L1$
   - a. Every element in $L2$ is appended to L - $|L1| \cdot \Theta(1) = \Theta(|L2|)$
   - b. Whole thing **takes $\boldsymbol{\Theta(|L2|)}$**

(d) Compute the length of a list: $len(L)$ **takes $\boldsymbol{\Theta(1)}$ time** (Python maintains a field with the list's length)

(e) $L.sort()$ **takes $\boldsymbol{\Theta(n \log n)}$ time**, where $n = |L|$

Python dictionaries or hash tables

(a) $D[key] = val$ **takes $\boldsymbol{\Theta(1)}$ time** with high probability.

(b) $key\ in\ D$ **takes $\boldsymbol{\Theta(1)}$ time** with high probability.

Document distance

We have two documents D_1 and D_2 and we want to compute the *distance* $d(D_1, D_2)$ between them.

We will define distance in terms of *shared words*. Let's look at two popular documents:

D_1: "the cat loves a cat"

D_2: "the dog loves another dog"

We define the distance as the angle between the vectors representing the word occurrences in the documents. These vectors have as many dimensions as the number of unique words in the two documents.

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

For our two documents, the sequence of unique words is “the, cat, dog, loves, another, a.” In this case, if we were to number the words from 0 to 5, the word vectors would be:

$$D_1 = [1,2,0,1,0,1]$$

$$D_2 = [1,0,2,1,1,0]$$

$$d(D_1, D_2) = \frac{1\cdot 1 + 2\cdot 0 + 0\cdot 2 + 1\cdot 1 + 0\cdot 1 + 1\cdot 0}{\sqrt{1\cdot 1 + 2\cdot 2 + 0\cdot 0 + 1\cdot 1 + 0\cdot 0 + 1\cdot 1}\cdot\sqrt{1\cdot 1 + 0\cdot 0 + 2\cdot 2 + 1\cdot 1 + 1\cdot 1 + 0\cdot 0}} = \cdots$$

Document distance algorithm in steps

- Split each document into words (parsing)
- Count the word frequencies
- Compute the distance as the dot product

Some notation:

- n - # of words
- $|word|$ = # of characters in word
- $|doc| = \sum|word|$ = # of characters in the document

(1) Splitting algorithm:
   a. For char in doc
      i. If not alphanumeric
         1. Add previous word (if any) to list and start new word
         2. Start new word

Note: You have to be careful how you add your words to your list. If the list has n words adding a new word using $L1 + [word]$ will take $\Theta(n)$ time instead of $\Theta(1)$.

- The complexity of the splitting code as result will take $\Theta(n^2)$, because the cost of adding word 1 through n will be $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = \Theta(n^2)$.
- If you use the $extend$ operation, each word is added in $\Theta(1)$ time and the splitting code will take $\Theta(n)$ time.