2. Models of Computation, Document Distance

Link: https://www.youtube.com/watch?v=0M_kIqhwbFo

The word algorithm is derived by al-Khwarizmi (father of algebra)

What's an algorithm?

- Computational procedure for solving a problem
- Input -> alg -> output
- Expressed in pseudcode and structured english instead of a programming language
- A model of comptuation compared to a computer for a program

Model of computation specifies:

- What operations an algorithm is allowed
- cost(time.. Of each application)

1.RAM:

- Random access machine in an algorithm
- Random access memory in a program
    - modeled by a big array with O(1) access times
    - Load O(1) words
    - Do O(1) computations
    - Store O(1) words
- O(1) registers
- Word is w bits
    - W log(size of memory)

2.Pointer Machine:

- Dynamically allocated objects
- Object has O(1) fields
- Field = words (e.g. int) or pointer to object or null
- Doubly linked list is a pointer machine

Python offers both pointer machines and references.

Python Model:

- "List" = array
- L[i] = L[j] + 5
- You can also do this in a object with O(1) attributes
    - X = x.next -> O(1) time
    - We don't naturally have linked list in Python
- Python performs table doubling for expanding the list size
    - It takes O(1) time

- For list concatenations in Python, it would take O(1 + |L1| + |L2|) time
- len(L) is constant time because there's a length property on the list
- L.sort() -> O(|L| lg |L|)
- D[key] = val -> O(1) [L8-10]
    - Dictionaries/Hash tables take constant time to access and delete values
        - With high probability due to hash collision
- Long:
    - x + y -> O(|l| + |y|)
    - X * y -> O((|x| + |y|)^lg3) lg3 is equal to 1.6
        - Optimized over grade school multiplication
- Heapq

Document Distance Problem:

- d(D1,D2)
- Document = sequence of words
- Word = string of alphanumeric chars.
- Idea: shared words
- Think of a document as a vector
- D[w] = # occurrences of w in D

Wrong Speculated Formulas:

D'(D1,D2) = D1 * D2

= **Σ D1[w]*D2[w]**

- The higher the product, the more commonality

d'(D1,D2) = D1 * D2/(D1 * D2)

d(D1,D2) = arcoss

Algorithm:

- 1) Split doc into words
- 2) Compute word frequencies
    - For word in doc:
        - Count[word] += 1
    - Linear time with good probability due to hashing
- 3) Compute dot product

* Avoid using re if you don't know the asymp complexity behind it