# Tree of Knowledge

*CS 106X Final Project*


Michael Sun

March 21 2019

This project attempts to recursively extract meaningful parent-child relations from a given high-level topic and insert them into a tree object of any depth. This facilitates understanding of high-level topic by breaking it down into perquisite subtopics.

# Sample Trees

A few extraneous topics from each tree has been removed so it can fit in the space below.

## "World" (depth = 3)

- World
  - History
    - Art
      - Century
      - Culture
      - Day
      - Present
    - Historian
      - Thucydides
    - Past
  - Human
    - Population
      - Animal
      - Human population
    - Ape
      - Apes
        - Hominid
      - Tool
    - Art
      - CIA
        - Culture
        - Language
      - Homo
      - Religion
      - Species
    - Homo sapiens

## "Computer Science" (depth = 3)

- Computer Science
  - Computation
  - Data
    - Concept
      - Knowledge
        - Idea
        - Philosophy
      - Number
        - Cognition
      - Abstract object
      - Mental representation
    - Information
      - Raw data
      - Bit
      - Code
      - Communication
        - Message
      - Concept
        - Data
          - Understanding
        - Knowledge
      - Entity
      - Sign
      - Uncertainty
  - Programming language
    - Computer
      - Arithmetic
      - Machine
      - Output device
      - Peripheral
      - Processing element

# Algorithm

C++ is a notoriously bad language for web-scraping, and my .pro file wasn't cooperating to my attempted adding of libcurl and other XML/JSON parsing libraries. Instead, I sticked with Python and parsed the following two function return values through a system command in C++.

Get Subtopics Function (topic):

> Iterates over all the links of the wikipedia page for the given topic
> Returns all those that appear more than once in given topic's page content

Parent and Child Function (topic1, topic2):

> Opens up the topic1 and topic2's (handling errors thereof) pages
> Counts the number of mentions of topic1 in topic2's page
>> Standardizes it by topic2's total content length
>> Stores into count1in2
> Counts the number of mentions of topic2 in topic1's page
>> Standardizes it by topic1's total content length
>> Stores into count2in1
> Determines
>> topic1 as parent if count1in2 > count2in1
>> topic2 as parent if count2in1 > count1in2
> Returns parent and relationship score, which is
> 100 x (max(c1in2, c2in1) - min(c1in2, c2in1)) / min(c1in2, c2in1)

The trade-off for using Python is that opening up and parsing Wikipedia pages are very slow, so is reading the buffers of system calls. Thus, the overall program runs slowly. I hope to improve the project after the class ends, by first converting the two algorithms into C++.

# Implementation Choices and Features

```
struct knode {
        string topic;
        Vector<knode *> children;
}
```

Taking advantage of the tree structure, I defined the following essential recursive data structure / functions:

void KnowledgeTree::fillTree(int level)
This is the main function implemented breadth-first

void KnowledgeTree::addToTree(knode *node, string topic, int curLevel)
This is a helper function that adds a topic

void KnowledgeTree::recursiveToString(int level, knode * startNode)
This recursively prints in a depth-first manner the entire tree by using indents to cout children under their parent.

void KnowledgeTree::mapToString()
This prints out all topics at a given level, the purpose being one can find all perquisite concepts at a given conceptual level, depending on how much in depth he/she wants to go into.

void KnowledgeTree::trainOfThought(string &cachedTrain, knode * startingNode)
This function prints all paths from the origin node to a leaf, in a complete train of thought.
(e.g. Computer Science->Data->Information->Communication->Message)