PFP Project Proposal - Trie AutoComplete

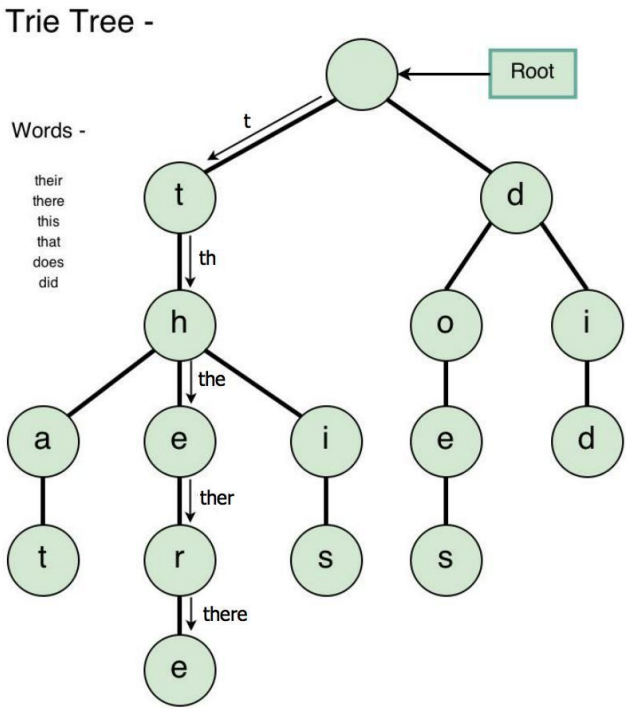
Siddharth Pittie - sp4013
Thang Nguyen - tn2468

Idea

Our project idea involves creating a word autocomplete feature, similar to what is commonly seen on search engines, where a word is partially typed and the suggestions are generated to complete the word. For example, typing "do" into Google may generate suggestions like "dog", "dominos", "docs", etc. Our project would aim to take in a string for a partial word and return *n* suggestions of the most likely completions for the string.

Design and Algorithm

The primary data structure needed for this project is a Trie (prefix tree). Tries are an ideal structure to represent words, this is visible from the image below -



Source: http://theoryofprogramming.azurewebsites.net/wp-content/uploads/2015/06/trie12.jpg

The trie would be in the form of a 26-ary tree, with each child node representing a letter in the alphabet, along with a boolean flag to represent whether the current node can be the end of a word, and an integer for the number of occurrences of the word.
Continuing with the example from earlier, if the string "do" is entered, the tree is traversed by first going to node "d" and then to its child "o", following which a depth-first-search is performed on every child tree that forms a word, and the word is added to a priority queue, along with its frequency. Then, the first *n* elements are popped from the priority queue and returned.

Data

The dataset we plan to use is a Wikipedia dataset, since it is a good representation of English text. This is crucial since we want to calculate and store word frequencies. We will parse this dataset and create the data structure described above. For calculating the word counts we will use a map reduce approach.

Parallelization

There will be 2 places in which we will take advantage of parallelization. Firstly, the map reduce task of calculating the frequency of each word in the dataset can be performed in parallel, since map reduce is ideal for such a task and should be immensely more performant than calculating the frequencies sequentially.
Secondly, when DFS is performed on the trie, each path can be traversed in parallel and the (word, frequency) pairs can be saved in memory. In theory, this would reduce the computational time of going through every path to just going down the longest path. Given a corpus of a large number of words, we expect this to give us significant performance benefits.

References

1. https://medium.com/analytics-vidhya/autocomplete-using-tries-42aadd875d72
2. https://www.geeksforgeeks.org/trie-insert-and-search/
3. https://dzone.com/articles/word-count-hello-word-program-in-mapreduce