Anagram Phrase Solver

Alexandre Laplante

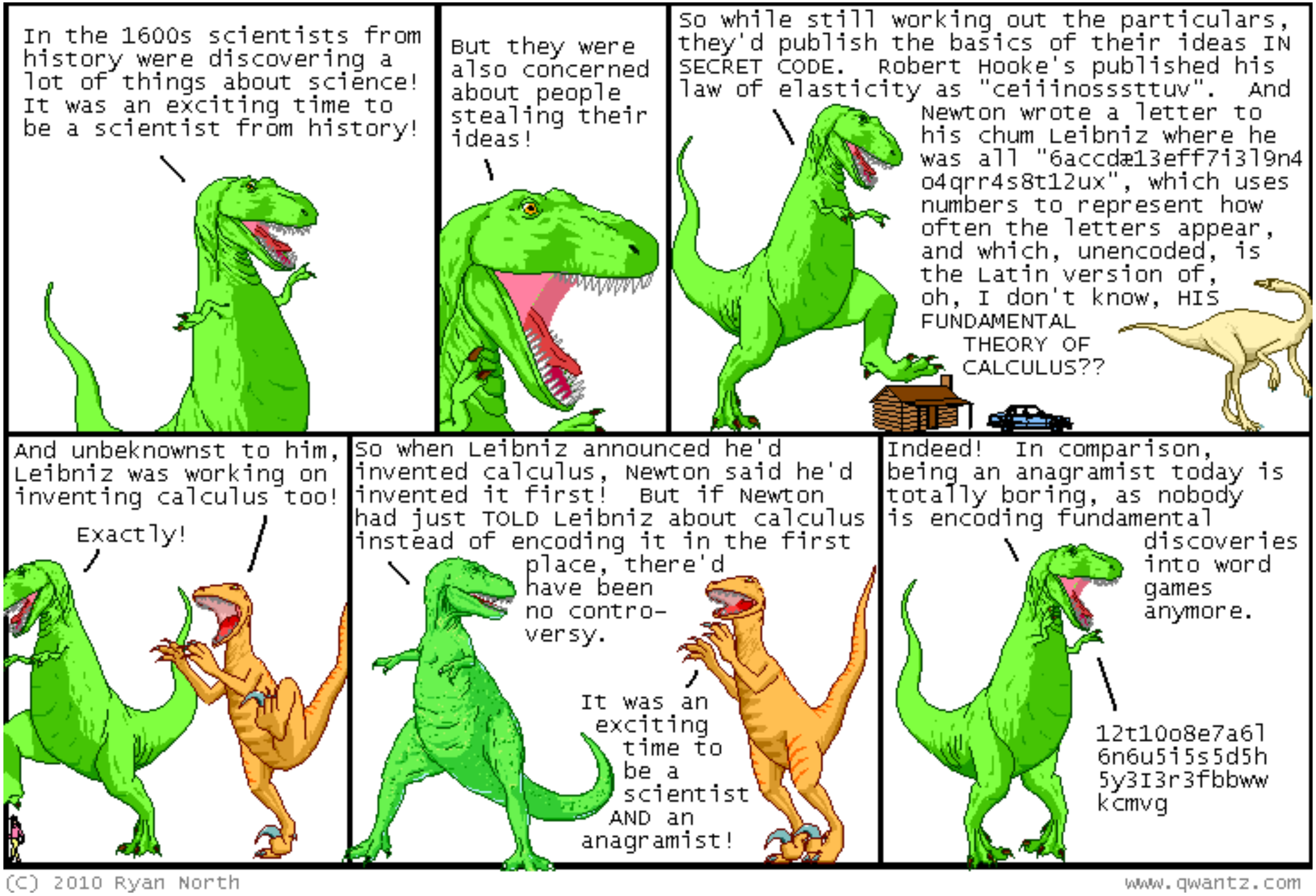
The Problem

Given a dictionary of words
ex: W=[the, it, they, chair, table, door]

Given a set of letters
ex: A="acehhirt"

Does there exist a phrase made out of the words in the dictionary W, such that every letter in A is used once?
ex: "the chair"

History


In the 1600s scientists from history were discovering a lot of things about science! It was an exciting time to be a scientist from history!
But they were also concerned about people stealing their ideas!
So while still working out the particulars, they'd publish the basics of their ideas IN SECRET CODE. Robert Hooke's published his law of elasticity as "ceiiinosssttuv". And Newton wrote a letter to his chum Leibniz where he was all "6accdæ13eff7i3l9n4o4qrr4s8t12ux", which uses numbers to represent how often the letters appear, and which, unencoded, is the Latin version of, oh, I don't know, HIS FUNDAMENTAL THEORY OF CALCULUS??
And unbeknownst to him, Leibniz was working on inventing calculus too!
Exactly!
So when Leibniz announced he'd invented calculus, Newton said he'd invented it first! But if Newton had just TOLD Leibniz about calculus instead of encoding it in the first place, there'd have been no controversy.
It was an exciting time to be a scientist AND an anagramist!
Indeed! In comparison, being an anagramist today is totally boring, as nobody is encoding fundamental discoveries into word games anymore.
12t10o8e7a6l6n6u5i5s5d5h5y3I3r3fbbwwkcmvg
(C) 2010 Ryan North
www.qwantz.com

History

Robert Hooke first published his famous law of elasticity now known as Hooke's law as: "ceiiinosssttuv". This is an anagram phrase in latin. The intended solution is "*Ut tensio, sic vis"*, meaning, "As the extension, so the force"[1]

Sir Isaac Newton's anagram phrase encoding the fundamental theorem of calculus was "6accdae13eff7i3l9n4o4qrr4s8t12ux", with the solution "Data aequatione quotcunque fluentes quantitates involvente, fluxiones invenire; et vice versa", meaning "Given an equation involving any number of fluent quantities to find the fluxions, and vice versa"[2]

[1] http://www.lindahall.org/events_exhib/exhibit/exhibits/civil/design.shtml

[2] http://www.mathpages.com/home/kmath414/kmath414.htm

Solution 1: Model as Integer Linear Program

Anacryptogram:

W=[aaa, bbb, abb]
A=[aaabbb]

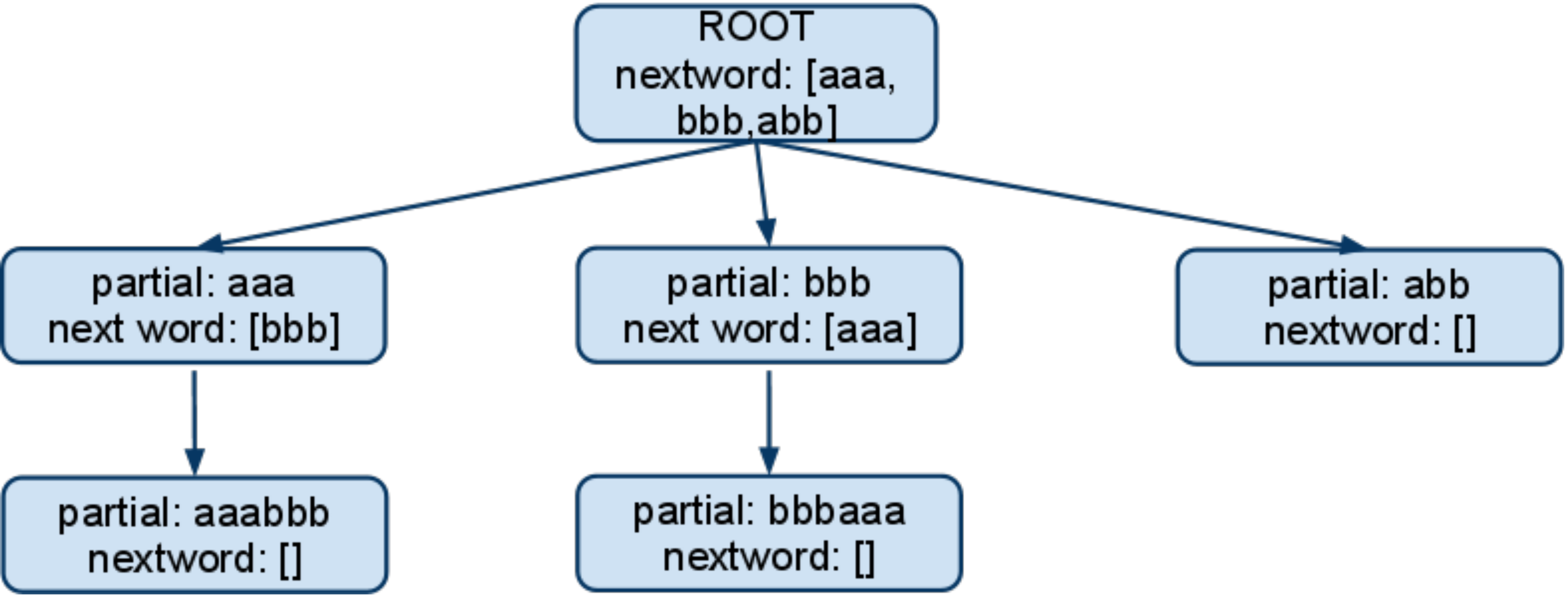
ILP:

```
max: a+b;

/* subject to */
a <= 3;
b <= 3;

w1a = 3 w1;
w1b = 0 w1;

w2a = 0 w2;
w2b = 3 w2;

w3a = 1 w3;
w3b = 2 w3;

a = w1a + w2a + w3a;
b = w1b + w2b + w3b;

int w1, w2, w3;
```

Solution 2: Backtracking Algorithm

Partial solutions consist of phrases that use at most the amount of letters given.

Example search tree:



Solution 2: Backtracking Algorithm

Needs:

- Efficient Subroutine to generate the next possible words
- Good Backtracking criterea

Solution 2: Backtracking Algorithm

To generate the next word, we store a data structure wordSet with a constructor wordSet(dictionary) and a method wordSet. get(letters[]).

The constructor takes the set of all words and creates the 3d array of Strings S[][][].
S[**i**][**j**] is the set of all words where the letter **i** appears at most **j** times.

The method get(letters[]) takes an array where letters[**i**] = the number of the letter **i** we have left to use on our partial solution.

Solution 2: Backtracking Algorithm

get(letters[]) returns the intersection of all S[i][letters[i]].

Remember: S[**i**][**j**] is the set of all words where the letter **i** appears at most **j** times.

Thus it returns the words for which we have enough letters.

This is accomplished by first finding out which of S[i][letters[i]] is smallest, then copying this set, then removing from this set all words that don't belong to EVERY other S[i][letters[i]], effectively taking the intersection of the sets.

Solution 2: Backtracking Algorithm

Pseudocode:

```
class wordSet
    constructor(words[])
        create the sets S[i][j] //this is long, but only happens once.

    get(letters[])
        min = min from i=0 to alphabetSize of (S[i][letters[i]])
        Set possibleWords = S[min][letters[min]]
        for(i = 0 to alphabetSize)
            if (i != min)
                for(w ∈ possibleWords)
                    if (w ∈ S[i][letters[i]]) then possibleWords -= w
```

Solution 2: Backtracking Algorithm

Backtracking conditions:

Main conditions:

- There are no next possible words.
- One of our letters does not appear in any next possible word.

Secondary conditions (hardly ever used, or don't ensure correct answer):

- No vowels in letters[]
- Not enough vowels in letters[]
- All possible next words put together are not enough to use all of our letters. (this is not valid if the last words are all the same word)

Solution 2: Backtracking Algorithm

A final improvement:

Only accept partial solutions in lexicographical order.

Modify wordSet.get(letters[], lastWord) not to return any words lexicographically greater than than the lastWord.

Sort the result of wordSet.get in order to traverse the search tree in lexicographical order.

This ensures that we don't repeat the same solution more than once.

Benchmarks

Hooke's law anagram with small latin dictionary
W = [ut, tensio, vis, sic, vir, sub, ter, tu, tui, tutis, totus, tot, ter, vae, vel, vere, via, vir, vito, vita, et, vice, versa, ac, is, ovis, te, tunc, iste, st, vinco, se, insto, intus, sui, sto, toties, viscus, incito, sive, us, in, sicut, vestis, os, si, victus, vesco, tui, nisi, vos, no]
A = "ceiiinosssttuv"

Solving the ILP

Using lp_solve:
Time to load data was 0.031 seconds, presolve used 0.000 seconds,
... 0.031 seconds in simplex solver, in total 0.062 seconds.

Backtracking

Time to find the first solution 0ms.

Time to find the last solution 141ms.

Benchmarks

Hooke's law anagram with medium latin dictionary (> 500 words)

Solving the ILP
Using lp_solve:

SUBMITTED
Model size: 13286
constraints, 13796
variables,

In the total iteration count
4399
Time to load data was 1.969
seconds, presolve used
0.031 seconds,
... 6.422 seconds in simplex
solver, in total 8.422 seconds.

Backtracking

Time to find the first solution
16ms.

Time to find the last solution
485ms.

Benchmarks

Hooke's law anagram with large latin dictionary (> 3000 words)

Solving the ILP
Using lp_solve:

SUBMITTED
Model size: 80808 constraints, 83915 variables,

In the total iteration count 37319,
Time to load data was 132.344 seconds, presolve used 0.172 seconds,
... 416.203 seconds in simplex solver, in total 548.719 seconds.

Backtracking

Time to find the first solution 32ms.

Time to find the last solution 12282ms.