A Modern spell(1)

Abhinav Upadhyay
<abhinav@NetBSD.org>
EuroBSDCon 2017, Paris

Outline

- ➢ Shortcomings in the old spell(1)
- ➢ Feature Requirements of a modern spell(1)
- ➢ Implementation Details of new spell(1)
- ➢ Performance comparison with other open source alternatives
- ➢ Integrations and demos

The beginning of the end

```
>Description:

spell(1) is a bit lacking. While it works on simple cases, e.g.
   valkyrie% echo 'frog' | /usr/bin/spell
   valkyrie% echo 'frogp' | /usr/bin/spell
   frogp
it accepts some interesting things:
   valkyrie% echo 'frogment' | /usr/bin/spell
   valkyrie% echo 'frogmental' | /usr/bin/spell
   valkyrie% echo 'froghood' | /usr/bin/spell
   valkyrie% echo 'frogship' | /usr/bin/spell
   valkyrie% echo 'biofrog' | /usr/bin/spell
   valkyrie% echo 'electrofrog' | /usr/bin/spell
   valkyrie% echo 'overfrog' | /usr/bin/spell

All hail the overfrog, or something.

This is because it has a set of suffix and prefix combining rules that
it applies rather ... liberally.

>How-To-Repeat:

>Fix:

I dunno. My inclination is towards cvs rm -- there are perfectly good
third-party spellcheckers at this point, natural language processing
is not exactly core OS functionality or the project's core competency,
and I don't think there's any need to maintain our own program given
that it doesn't work very well.
```

Shortcomings in the old spell(1)

- Very old - dates back to Unix Version 7

Shortcomings in the old spell(1)

- ➢ Very old - dates back to Unix Version 7
- ➢ Uses inflection rules

Shortcomings in the old spell(1)

- Very old - dates back to Unix Version 7
- Uses inflection rules
    - First checks if a word exists in the dictionary or not
    - If it does not -
    - Checks if the string contains certain prefixes - (pre, post, anti, meta, non, re) and removes them
    - Checks if the string contains certain suffixes - (ness, ed, ing, able, ly) and removes them
    - If final word exists in the dictionary, it believes spelling is correct

Shortcomings in the old spell(1)

- Very old - dates back to Unix Version 7
- Uses inflection rules
  - First checks if a word exists in the dictionary or not
  - If it does not -
  - Checks if the string contains certain prefixes - (pre, post, anti, meta, non, re) and removes them
  - Checks if the string contains certain suffixes - (ness, ed, ing, able, ly) and removes them
  - If final word exists in the dictionary, it believes spelling is correct
- The rules only apply for English language

Shortcomings in the old spell(1)

- Very old - dates back to Unix Version 7
- Uses inflection rules
  - First checks if a word exists in the dictionary or not
  - If it does not -
  - Checks if the string contains certain prefixes - (pre, post, anti, meta, non, re) and removes them
  - Checks if the string contains certain suffixes - (ness, ed, ing, able, ly) and removes them
  - If final word exists in the dictionary, it believes spelling is correct
- The rules only apply for English language
- No spelling corrections

Shortcomings in the old spell(1)

- Very old - dates back to Unix Version 7
- Uses inflection rules
  - First checks if a word exists in the dictionary or not
  - Checks if the string contains certain prefixes - (pre, post, anti, meta, non, re) and removes them
  - Checks if the string contains certain suffixes - (ness, ed, ing, able, ly) and removes them
  - If final word exists in the dictionary, it believes spelling is correct
- The rules only apply for English language
- No spelling corrections
- Lack of a library interface for other applications

Expectations from new spell(1)

Expectations from new spell(1)

- Do spell suggestions apart from just spell check

Expectations from new spell(1)

- ➢ Do spell suggestions apart from just spell check
- ➢ Not use algorithms strictly tied to just the English language

Expectations from new spell(1)

- ➢ Do spell suggestions apart from just spell check
- ➢ Not use algorithms strictly tied to just the English language
- ➢ Provide a library interface

What have I done?

What have I done?

- New bigger dictionary
- New spell(1) implementation using levenshtein distance, Double Metaphone algorithms, and ternary tries
- A benchmark comparison against aspell, ispell and hunspell
- Integration with sh(1) for auto-completion and spell check

New Dictionary

- Expanded /usr/share/dict/words
  - Includes all verb, noun and adjective forms

New Dictionary

- ➢ Expanded /usr/share/dict/words
    - Includes all verb, noun and adjective forms

	Old dictionary	**New Dictionary**
Size	235008	2.4M
Number of words	421128	4.5M

New spell(1) Implementation

- Two types of spell check problems

New spell(1) Implementation

- Two types of spell check problems
    - Non-word errors - e.g. *eppled* for *applied*

New spell(1) Implementation

- Two types of spell check problems
    - Non-word errors - .e.g *appled* for *applied*
    - Real-word errors - e.g. *dessert* for *desert*, *there* for *three, piece* for *peace*

Handling Real-word Errors

Handling Real-word Errors

- Much harder problem
- Cannot simply lookup the dictionary
- Word bi-grams or tri-grams could be used to detect real-word errors
  - *Apple feel from the tree*
  - *"feel"* not commonly used with "*apple*" and "*from"*, but "*fell"* is
- Much expensive, need to scan every word with a window of 3 or 4 words.
- Not in the scope of the current project but possible future work

Handling non-word errors

Handling non-word errors

- Very simple to detect (just look up the dictionary)

Handling non-word errors

- ➢ Very simple to detect (just look up the dictionary)
- ➢ No need for complex inflection rules with the expanded dictionary - much more reliable in detecting errors

Dictionary Representation and Lookup

Dictionary Representation and Lookup

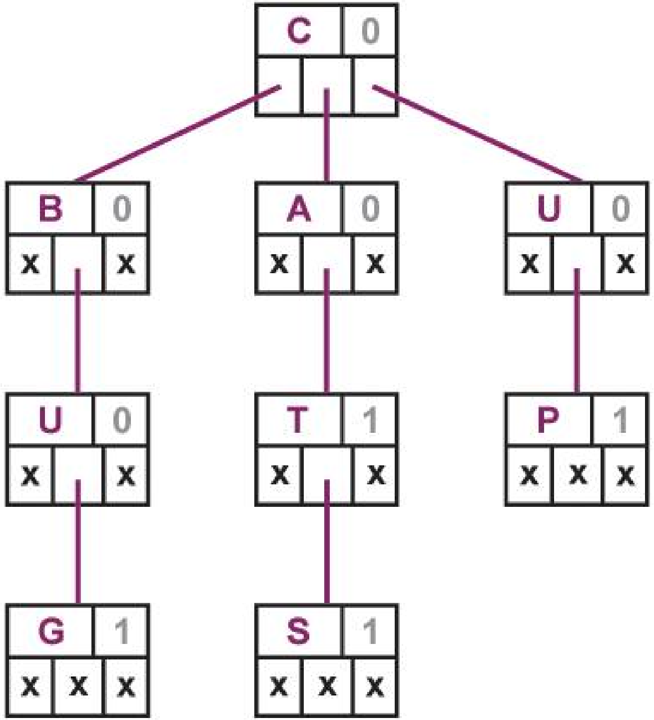
- ➢ Dictionary Representation - several options
- ➢ Hash table - O(1) lookup but no worse case guarantee
- ➢ Red Black Trees - O(lg n) guaranteed lookup time but requires complete string comparisons in the worst case
- ➢ Ternary Tries - O(lg n) lookup and does not require string comparisons with every word in the dictionary, but costs some extra memory

Ternary Search Tries

Ternary Search Tries

- ➢ Much like a binary search tree
- ➢ Each node stores one character and has three children (left, middle, right)
- ➢ Left subtree - for characters smaller than the character at the root node
- ➢ Right subtree - for characters greater than the character at the root node
- ➢ Middle subtree - for characters matching the character at the root node
- ➢ Provides symbol table APIs as well as APIs for prefix match

Ternary Search Tries



Ternary Search Tree for CAT, BUG, CATS, UP

Doing Spell Correction

Doing Spell Correction

- Edit Distance Technique
- Metaphone algorithm
- N-gram models

Edit Distance Techniques

- Edit distance - number of edits (insertion, deletion, replacement of characters) required in a word to convert into another word.

Edit Distance Techniques

- Edit distance - number of edits (insertion, deletion, replacement of characters) required in a word to convert into another word.
- A majority of spelling errors are just one 1 edit distance away from the correct spelling

Edit Distance Technique

Example of words 1 edit distance away from “*teh*”:

deletes = ['eh', 'th', 'te']

transpose = ['eth', 'the']

replaces = ['aeh', 'beh', 'ceh', 'deh', 'eeh', 'feh', ..., 'tez’]

inserts = ['ateh', 'bteh', 'cteh', 'dteh', 'eteh', 'fteh', ..., 'zteh']

Metaphone Algorithm

- A phonetic algorithm (a better replacement for soundex)
- Developed by Lawrence Phillips in 1990
- Superseded by Double Metaphone in 2000 (by the same author)
- Latest version Metaphone 3 (but only available as a commercial implementation)
- 99% accurate for English and covers peculiarities in several other languages as well (Slavic, German, Celtic, Greek, French etc.)
- Double Metaphone is used by aspell

Word Bigrams

Word Bigrams

- A useful technique to get more accurate suggestions
- When having more than possible corrections for a misspelled word -
- Look at the next and previous word and see which correction fits the best
- For instance: “*I am not feeling wery well*”

Strategy for Spell Correction

Strategy for Spell Correction

- ➢ Find all possible corrections at distance 1
- ➢ If no match found, find words having the same metaphone codes at distance 0, 1 and 2 with the misspelled word
- ➢ If still no match found, find words at edit distance 2

Strategy for Spell Correction

- Some tricks for improving accuracy:
  - Lower weight to candidate corrections requiring modification at first character
  - Lower weight to candidate corrections involving replacement of characters
  - Higher weight to candidates having same metaphone code as the original incorrect spelling

Performance Comparison

Performance Comparison

	First	1-5	1-10	1-25
Aspell 0.60.6/Normal	73.8	96.1	97.6	98.3
Aspell 0.60.6/Slow	74.0	96.6	98.2	99.0
Hunspell 1.1.12	80.5	96.5	97.1	97.1
ISpell 3.1.20	77.0	84.7	85.0	85.1
nbspell/slow	91.0	95.1	95.4	95.4
nbspell/fast	88.7	93.1	93.2	93.4

Demo

Conclusion

- Performance comparable to other popular open source implementations
- Much room for further investigation and improvement
- But nice to have a BSD licensed spell checker + library when you need it

Code

https://github.com/abhinav-upadhyay/nbspell

Questions

Thank you!