Groonga 改良型Ngram トークナイザー

Naoya (@naoa_y)

全文検索エンジンGroongaを囲む夕べ5
2014/11/29

自己紹介

- Naoya (@naoa_y)
  - 数年ほど特許事務所勤務
  - 前は数年ほどユーザSIでインフラSE
  - Groongaでプログラミングを学ぶ
  - GroongaのCプラグインなら書ける

制作物

- 特許の全文検索サービス Patent Field を個人で制作(中)
  - 専門家以外でも有用な知財情報へ迅速にアクセスできるように
  - 権利の死活情報でも絞込みができ侵害調査やフリーな技術調査が可能
  - 知財流通促進・フリーな技術流用による産業の発達促進
  - 今回紹介する改良もフリーな技術情報をヒントに発想を取り入れたもの

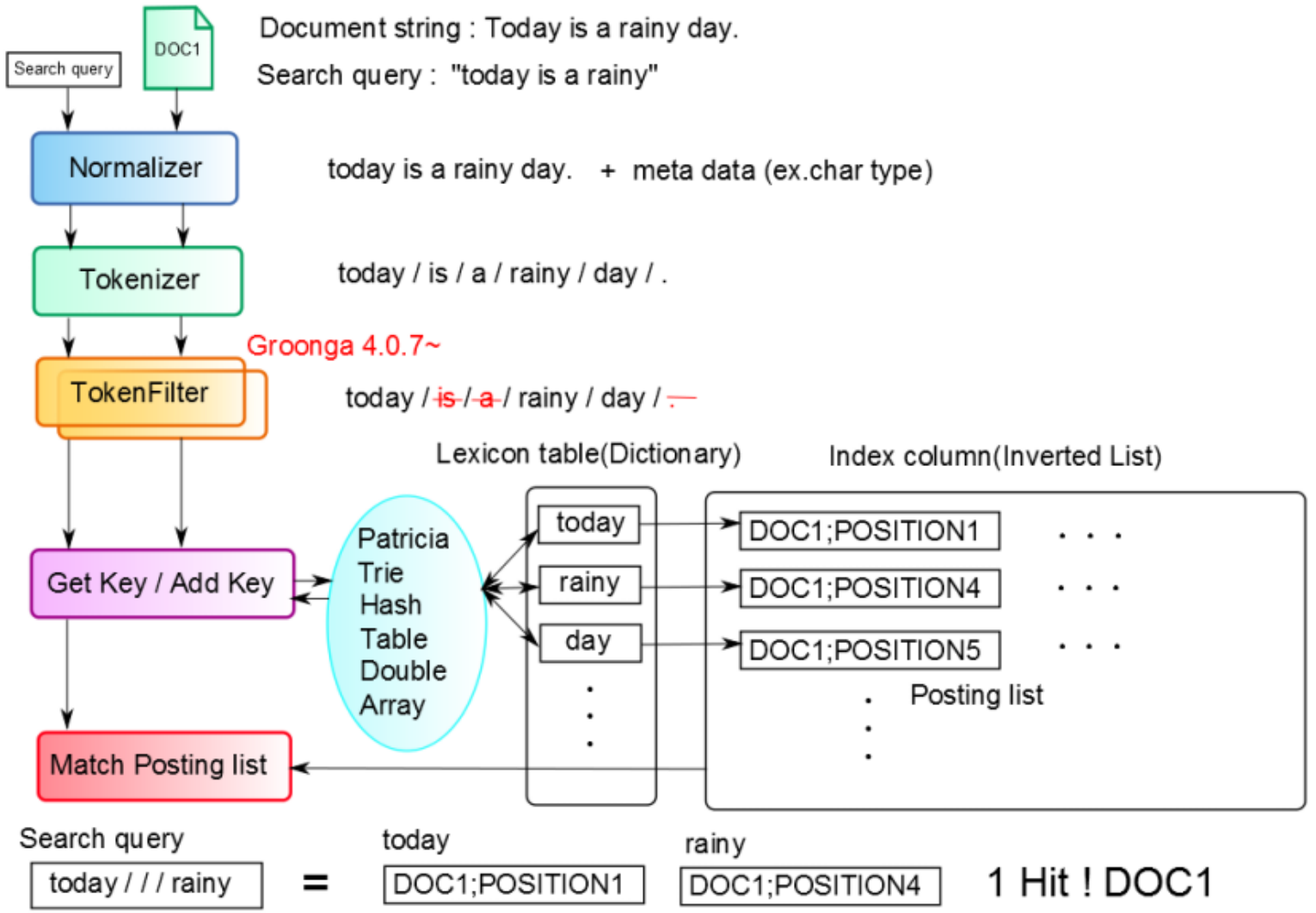
制作物

- ✓ 一番大きな**日本語**のデータベースで数百GiB超(カラム非圧縮)
- ✓ 数百GiB規模のDBを小規模でできるだけ実用的な速度で検索したい
- ✓ Ngramトークナイザーを高速化、効率化したプラグイン作成
  (したけど時間が足りなくてまだ反映できてない)

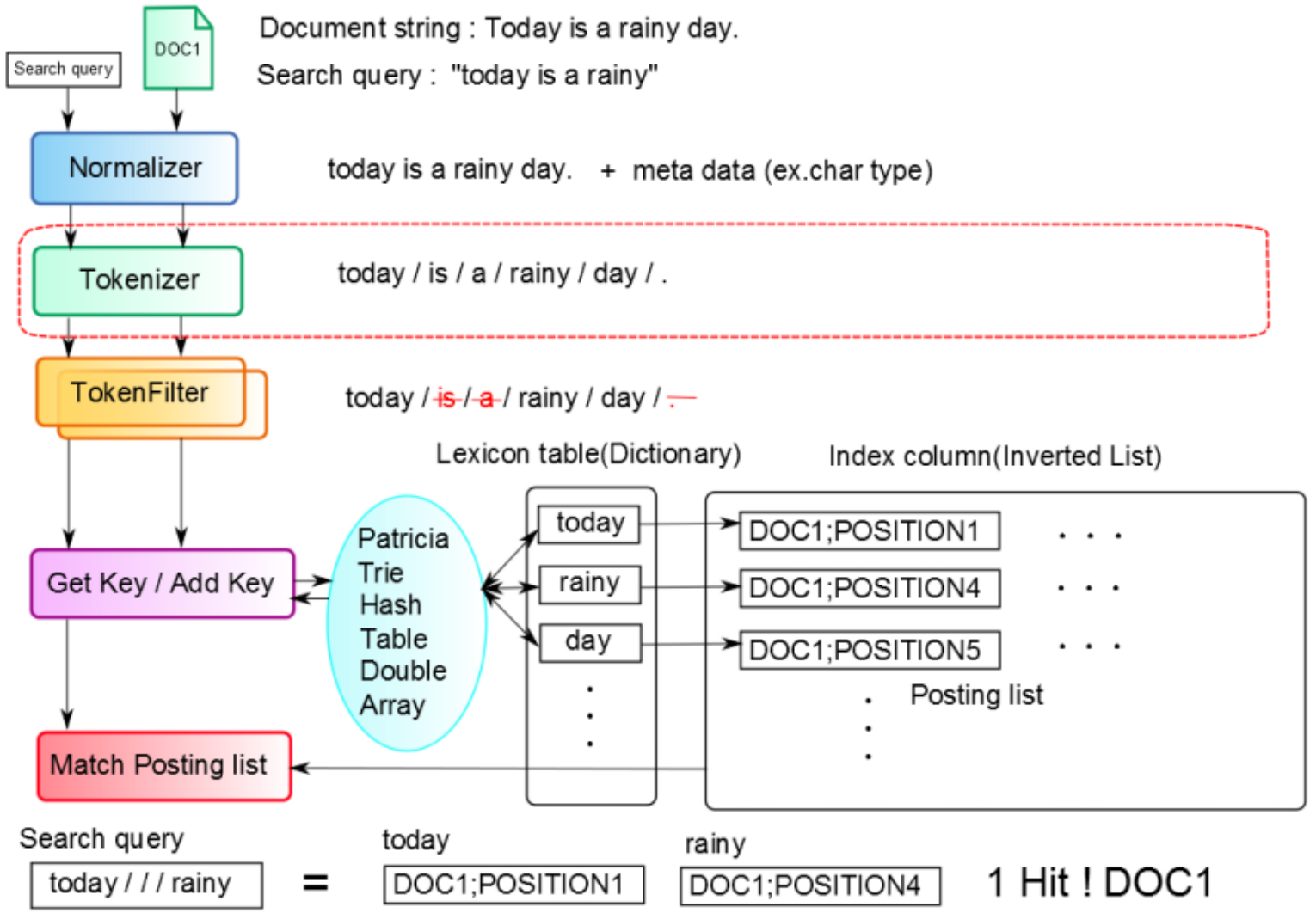
改良型Ngramトークナイザー

- YaNgram - Yet another Ngram tokenizer plugin
  https://github.com/naoa/groonga-tokenizer-yangram
- 検索時のオーバラップスキップ
- 静的な頻度情報に応じた可変Ngram(Vgram)
- 既知フレーズのグループ化

Groongaの全文検索の流れ



Groongaのトークナイザー



Ngramトークナイザー

- 所定の長さのユニットサイズで1文字ずつずらす
  1:Unigram 2:Bigram
  3:Trigram
- 例：「今日は雨だ」⇒
  「今日/日は/は雨/雨だ/**だ**」
- Groongaは1文字でも検索できるように末尾1文字も含まれる

Ngramトークナイザー

- デフォルトではアルファベット、記号、数字はグループ化
  - 検索ノイズ低減、検索速度向上のため
- アルファベット、記号、数字もNgramにしたいのであれば、TokenBigramSplit〜系を使う

Ngramトークナイザーのメリット

- ✓ 漏れのない検索が可能
- ✓ 新語、造語に対応
  特許文献の場合、権利解釈の範囲を狭めないように固有名詞があまり使われず商標も使えないので造語だらけ
- ✓ メンテナンスコスト不要

Ngramトークナイザーのデメリット

- 1文字ずつずらすためトークンの総数が多くなり転置索引のサイズが大きくなる
  転置索引のサイズはほぼトークンの総数によって決まる
- 検索ノイズが含まれることがある 例：東京都に対して京都でヒットする
- 日本語の場合、あまり大きな影響ではない 要件による

形態素解析トークナイザー

- 形態素解析器を使って文脈に応じて単語単位に分かち書き
  TokenMecab
- 分割ルールは学習モデルと辞書による Unidicであれば短く分かち書き
- 例：「今日は雨だ」⇒
  「今日/は/雨/だ」

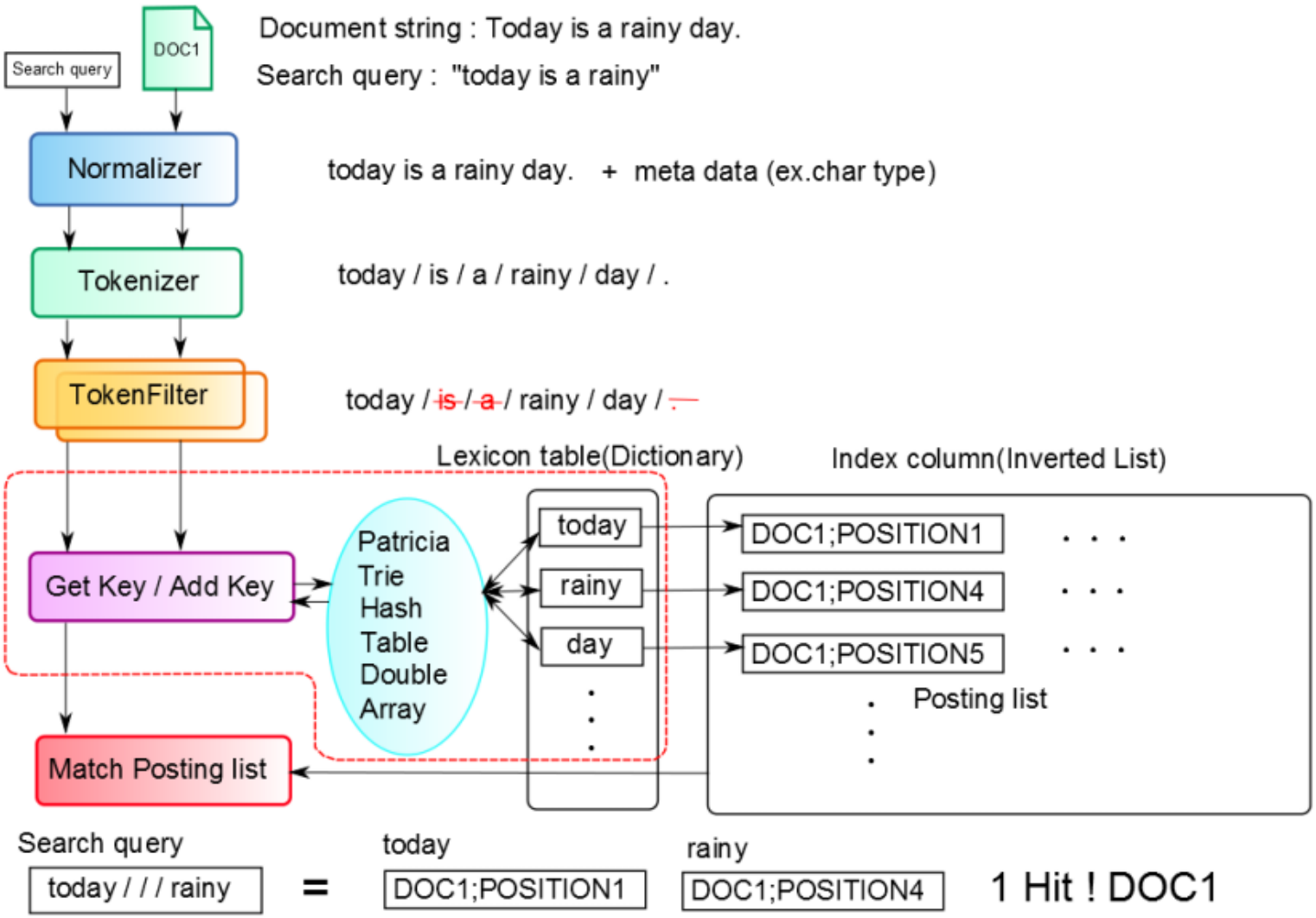
形態素解析トークナイザーのメリット

- ✓ 検索ノイズの低減
  例：東京都に対して京都がヒットしない
- ✓ 単語ごとにずらせるため転置索引のサイズがコンパクト
  形態素解析の場合「転置索引」⇒「転置索引」1つ
  Bigramの場合「転置索引」⇒「転置/置索/索引/引」4つ

形態素解析トークナイザーのデメリット

- ✓ 検索漏れ有
- ✓ 辞書の追加やモデルの再学習などメンテナンスコスト大
- ✓ 検索クエリと文章中では文脈が異なり分割ルールが異なることがまれによくある チューニングが大変

キー探索


Search query
DOC1
Document string : Today is a rainy day.
Search query : "today is a rainy"
Normalizer
today is a rainy day. + meta data (ex.char type)
Tokenizer
today / is / a / rainy / day / .
TokenFilter
today / is / a / rainy / day / .
Lexicon table(Dictionary)
Index column(Inverted List)
Get Key / Add Key
Patricia Trie Hash Table Double Array
today
rainy
day
DOC1;POSITION1
DOC1;POSITION4
DOC1;POSITION5
Posting list
Match Posting list
Search query
today / / / rainy
=
today
DOC1;POSITION1
rainy
DOC1;POSITION4
1 Hit ! DOC1

キー探索

- ハッシュ表やパトリシアトライなどを使って語彙表のキーとして登録されたトークンを探す
- いわゆる辞書引き・KVS
- Groongaではインデックス≠キー
- キー探索は非常に速くμsecオーダー KVSが速いのは知っているはず

キーの種類数が増える要因

- ✓ 文字の種類数が多いこと
  組み合わせが増えるためキーの種類数は多くなる
- ✓ 日本語の文字の種類は多い
  ひらがなカタカナ50種 漢字いっぱい
- ✓ 英語の文字の種類は非常に少ない
  アルファベット26種
- ✓ 日本語はキーの種類が多い

キーの種類数が増える要因

- ✓ 文字数が多いこと
  組み合わせが増えるためキーの種類数は多くなる
- ✓ NgramのNが大きいほどキーの種類数が多い

キーの種類数増によるキー探索速度への影響

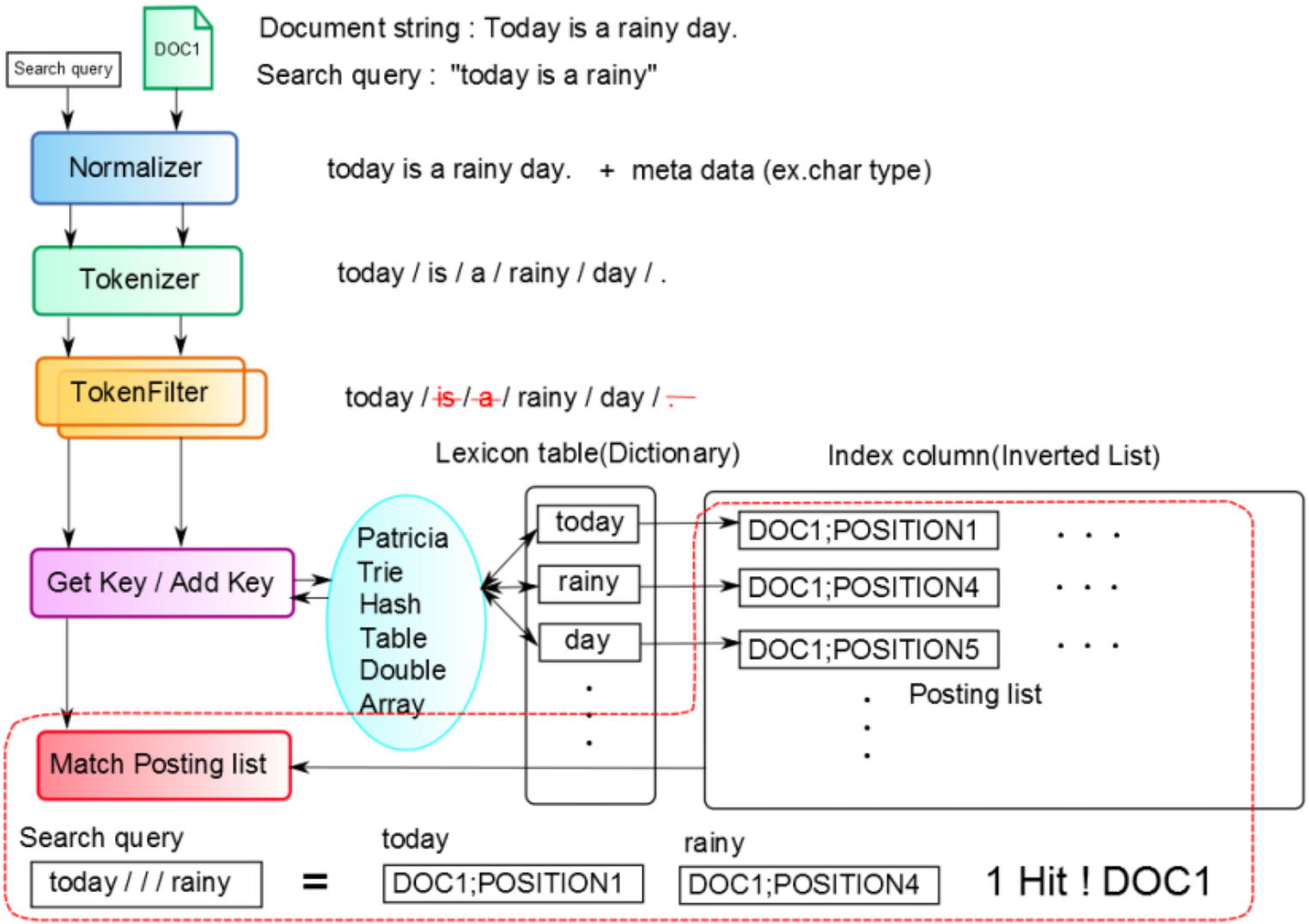
パトリシアトライ(ADD後⇒GET)

キー種類数	キー1件取得秒数
1万	21 μsec
1千万	37 μsec

キーの種類数増によるキー探索速度への影響

- キー探索はキーの種類が増えても線形的に時間が増えない
  例：ハッシュ表O(1)、パトリシアトライO(k)
- キー種類増の検索速度への影響は非常に軽微

ポスティング探索


Search query
DOC1
Document string : Today is a rainy day.
Search query : "today is a rainy"
Normalizer
today is a rainy day. + meta data (ex.char type)
Tokenizer
today / is / a / rainy / day / .
TokenFilter
today / is / a / rainy / day / .
Lexicon table(Dictionary)
Index column(Inverted List)
Get Key / Add Key
Patricia Trie Hash Table Double Array
today
rainy
day
DOC1;POSITION1
DOC1;POSITION4
DOC1;POSITION5
Posting list
Match Posting list
Search query
today / / / rainy
=
today
DOC1;POSITION1
rainy
DOC1;POSITION4
1 Hit ! DOC1

ポスティング探索

- キー探索によって取得したポスティングリスト中のトークンの出現位置と検索クエリのトークンの出現位置の相対的な並びが一致するかどうかを比較
- トークンの出現頻度が増えるとポスティングリストが長くなる
  - 一番時間がかかるところ
    シーケンシャルサーチを除く

トークンの出現頻度が増える要因

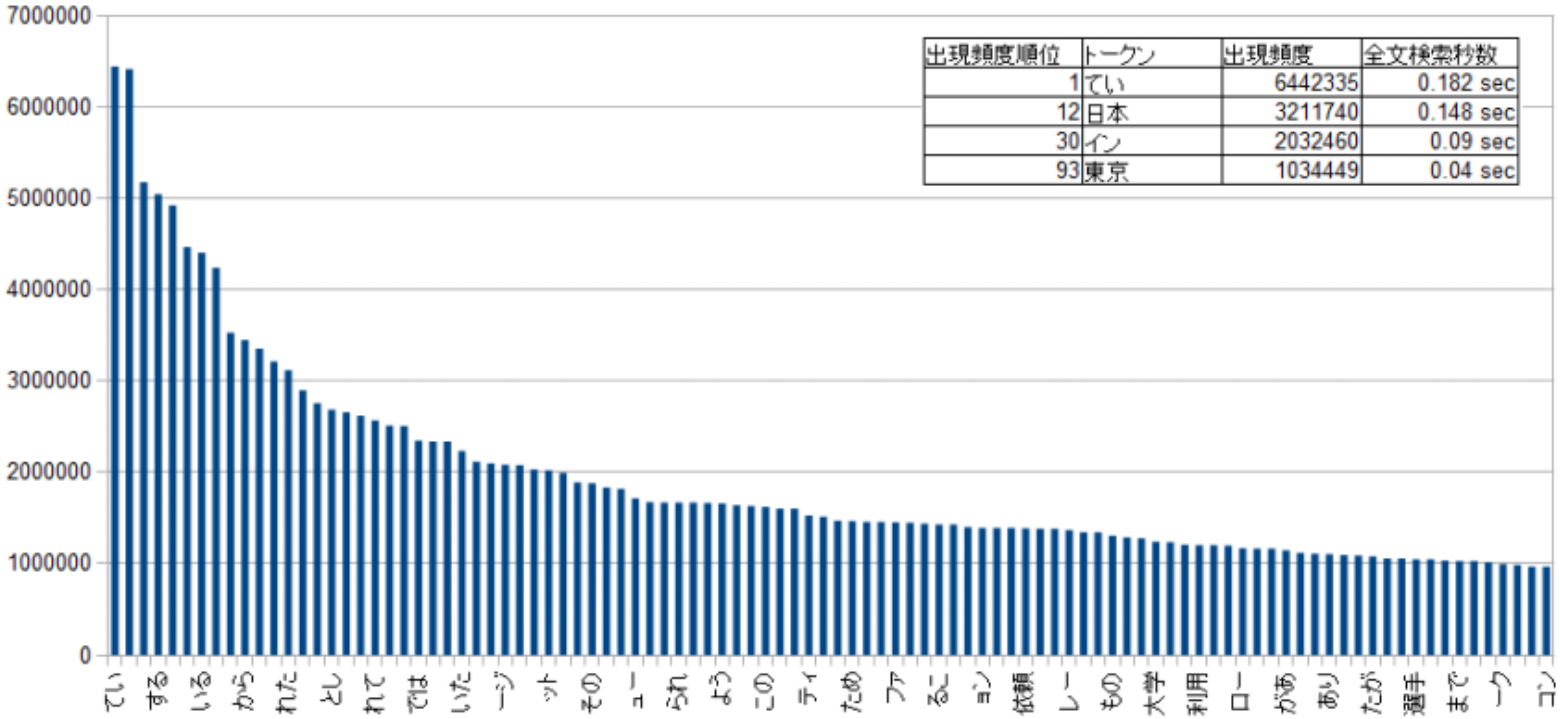
- 文字の種類数が少ないほどキーの種類数は少ない
  ⇒ トークン1個あたりの出現頻度は大きい
- 日本語はキーの種類が多い
  ⇒トークンの出現頻度が少ない

トークンの出現頻度が増える要因

- 文字数が少ないほどキーの種類数は少ない
  ⇒ トークン1個あたりの出現頻度は大きい
- NgramのNが大きいとキーの種類数が多い
  ⇒ トークンの出現頻度が少ない

Bigramトークナイザーの出現頻度と検索速度例

Wikipedia(ja)の TokenBigram によるトークンの出現頻度上位 100 位 (全体 2,382,473)



出現頻度順位	トークン	出現頻度	全文検索秒数
1	てい	6442335	0.182 sec
12	日本	3211740	0.148 sec
30	イン	2032460	0.09 sec
93	東京	1034449	0.04 sec

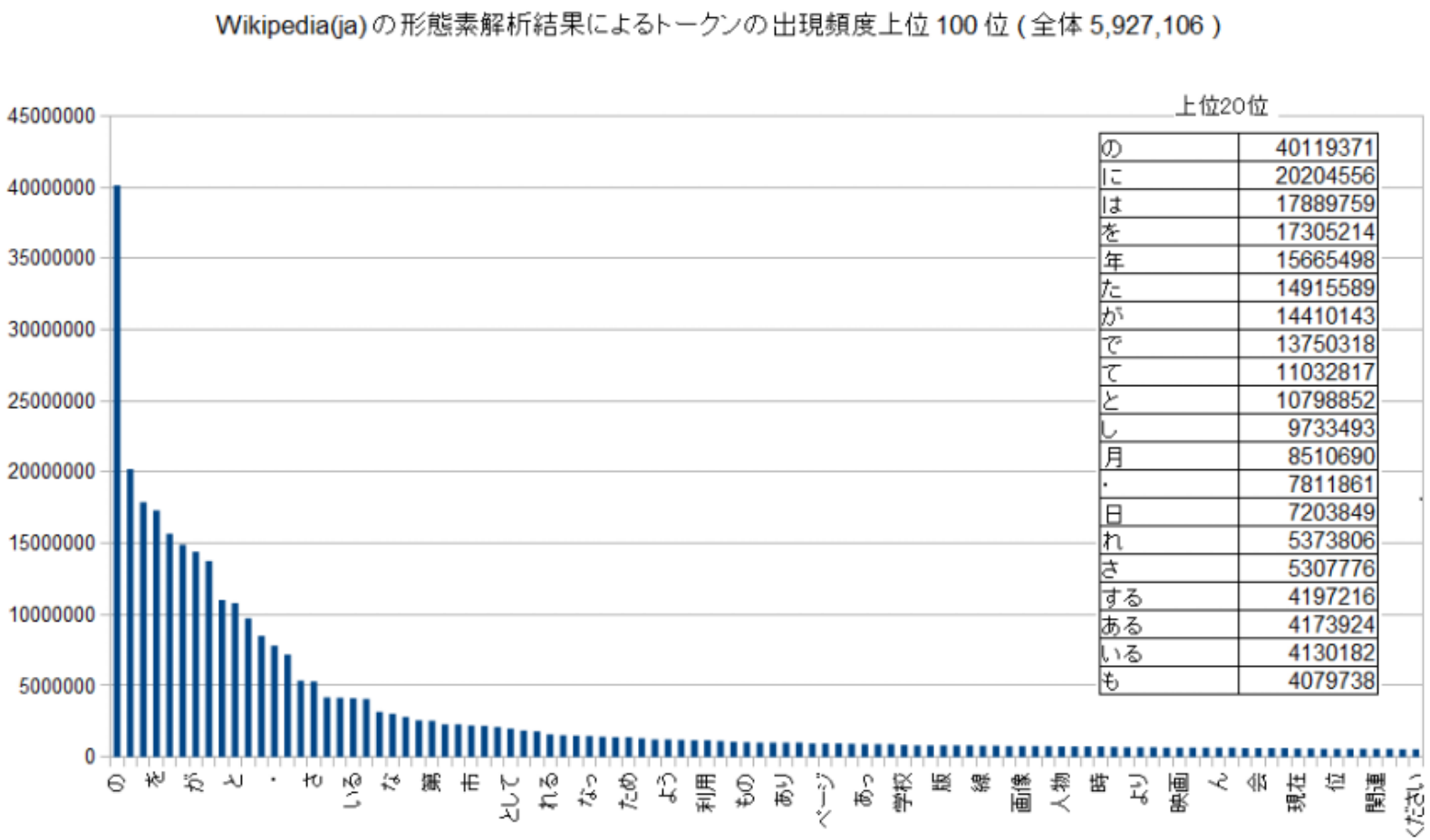
トークンの出現頻度増によるポスティング探索速度への影響

- 文書数/サイズが多くなるとトークンの出現頻度が増えポスティングリストが非常に長くなる
- トークンの出現頻度に応じてほぼ線形的に検索時間が伸びる
  - 大抵の場合、ポスティングリストの探索でCPUがボトルネック

検索速度を高速に保つために重要なこと

- キーの種類数よりもポスティングリストが長くなりすぎないようにする
- CPUクロック数を上げる

形態素解析トークナイザーの出現頻度例



形態素解析トークナイザーの高速化

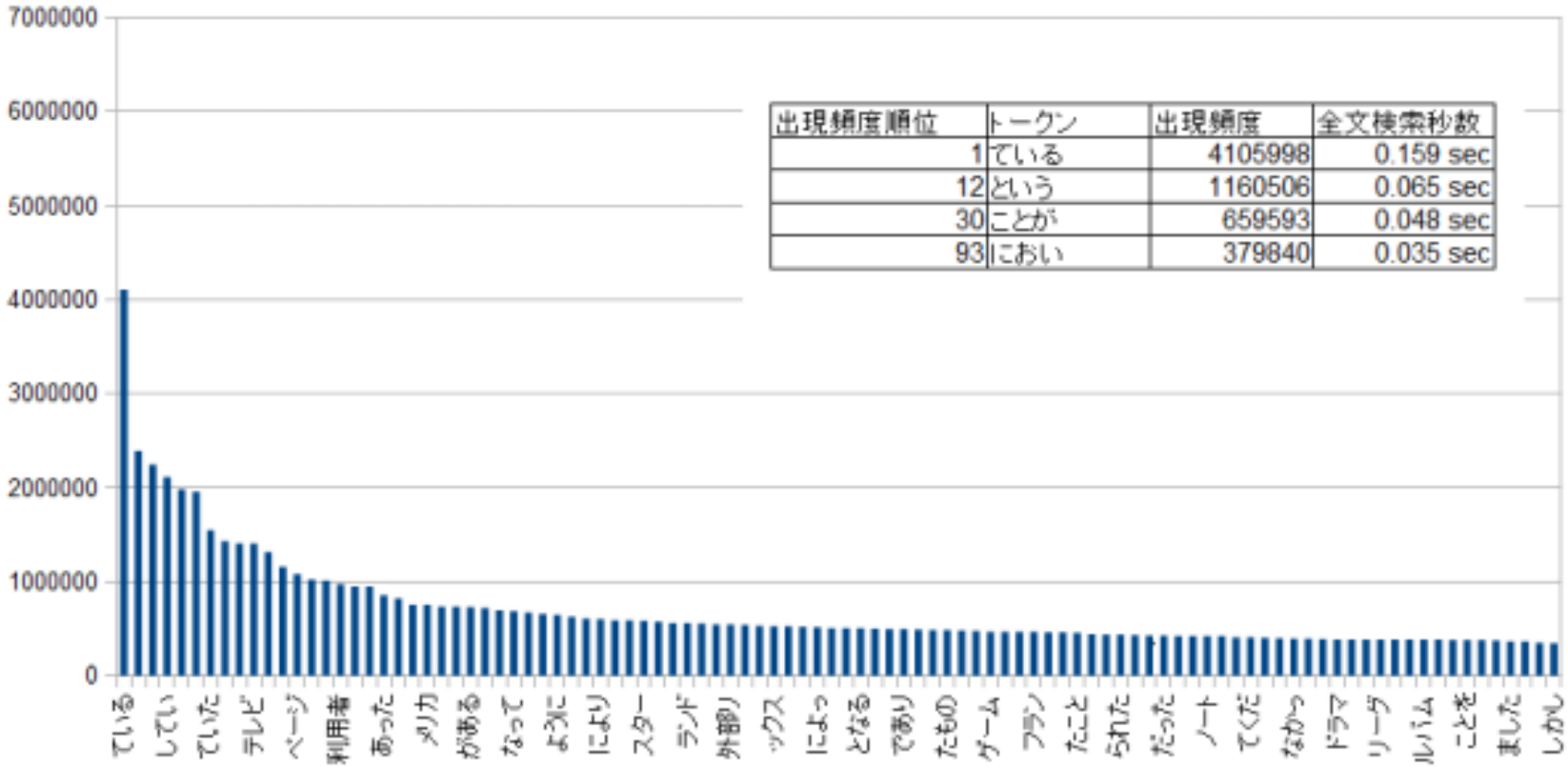
- ✓ 助詞などの頻出語をストップワードにする
- ✓ 頻出語を含む複合語を辞書登録
- ✓ 見出しタグ等文書に必ず含まれるフレーズを除去もしくは辞書登録

Ngramトークナイザーの高速化

- Nのサイズを大きくする
  Bigram ⇒ Trigram
- トークンの種類が増えて１つごとのポスティングリストは短くなる

Trigramトークナイザーの出現頻度と検索速度例

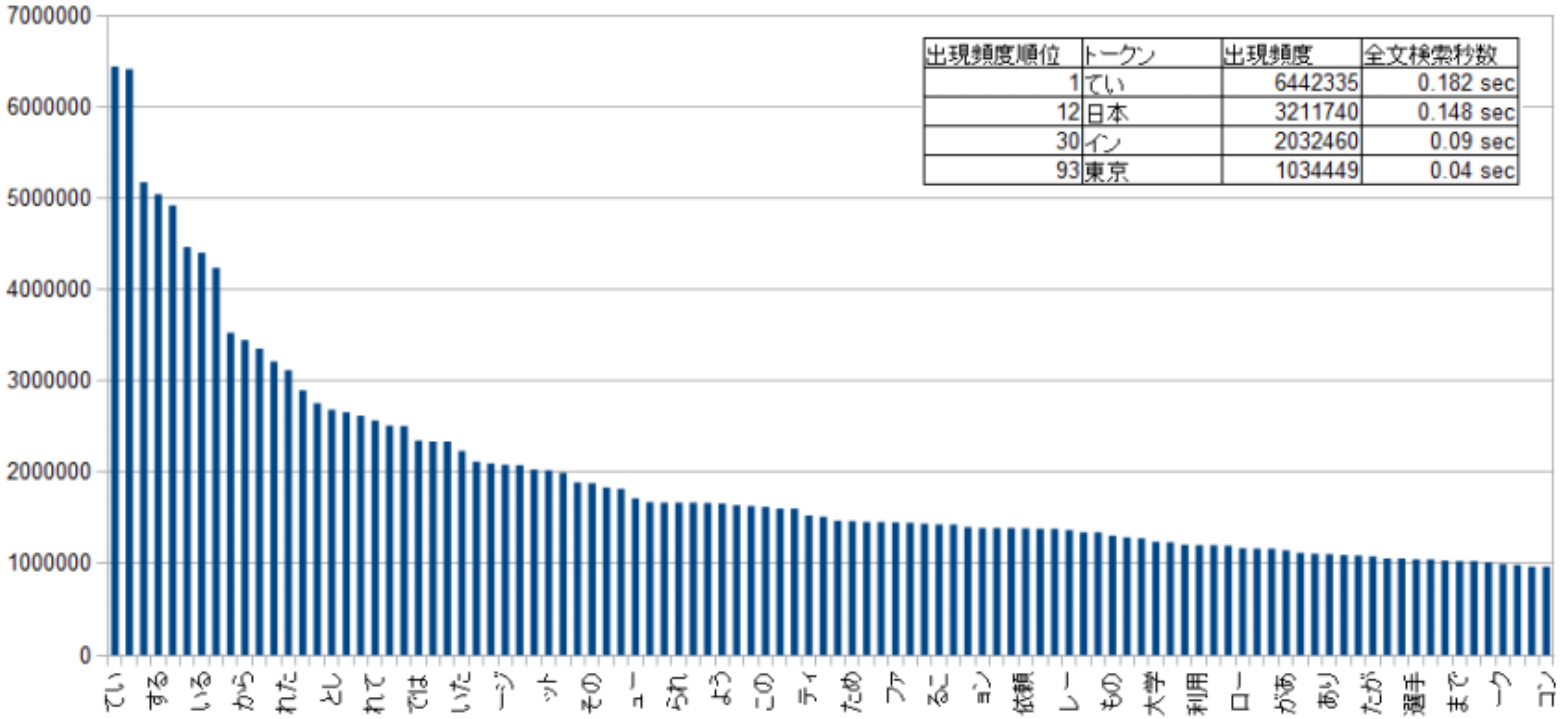
Wikipedia(ja) の TokenTrigram によるトークンの出現頻度上位 100 位 (全体 24,462,376)



出現頻度順位	トークン	出現頻度	全文検索秒数
1	ている	4105998	0.159 sec
12	という	1160506	0.065 sec
30	ことが	659593	0.048 sec
93	におい	379840	0.035 sec

Bigramトークナイザーの出現頻度と検索速度例(再褐)

Wikipedia(ja)の TokenBigram によるトークンの出現頻度上位 100 位 (全体 2,382,473)



出現頻度順位	トークン	出現頻度	全文検索秒数
1	てい	6442335	0.182 sec
12	日本	3211740	0.148 sec
30	イン	2032460	0.09 sec
93	東京	1034449	0.04 sec

Ngramトークナイザーの高速化

- Trigramにすれば基本的に3文字以上の検索速度が速くなる
- 日本語は文字種も多いためTrigramであればかなり速い
- が、まだ速くする方法がある

Ngramトークナイザーをさらに高速化するために

- Ngramの文書追加時は1文字ずらしでキーを登録する必要あ
  - 日本語は文章中の単語境界が判断できないため
  - 「今日は雨だ」⇒
    「今日/日は/は雨/雨だ/だ」

Ngramトークナイザーをさらに高速化するために

- 検索時は開始位置が決まっているので1文字ずらしする必要はない
  ⇒オーバラップ部分をスキップ
  - 「今日は雨」⇒「今日/日は/は雨」
    ×
  - 「今日は雨」⇒「今日/　　/は雨」
    ○

Ngramトークナイザーをさらに高速化するために

- ✓ 末尾で短くなるところは短い奴を採用するのではなく1つ手前の長い方を採用する
  短いやつはポスティングリストが長く検索が遅い
- ✓ 「今日は雨だ」
  ⇒「今日/　　/は雨/　/だ」✕
  ⇒「今日/　　/は雨/雨だ」◯

Ngramトークナイザーをさらに高速化するために

- これを追加実装したのが以下のトークナイザー

  TokenYaBigram
  TokenYaTrigram
  ~SplitSymbolAlphaもあり

TokenBigram/ TokenYaBigramの速度

Wikipedia(ja)で1000回検索

トークナイザー	検索秒数平均
TokenBigram	0.0508sec
TokenYaBigram	0.0325sec

TokenTrigram/ TokenYaTrigramの速度

Wikipedia(ja)で1000回検索

トークナイザー	検索秒数平均
TokenTrigram	0.0146sec
TokenYaTrigram	0.0063sec

TokenYaBigram/TokenYaTrigramの速度

- YaBigramはBigramに比べ1.5倍ほど速い
- YaTrigramはTrigramに対して2倍ほど速い
- NgramのNのサイズが大きいほどオーバーラップを飛ばす量が大きくなるためより速くなる

TokenBigram/ TokenTrigramのキー

Wikipedia(ja)

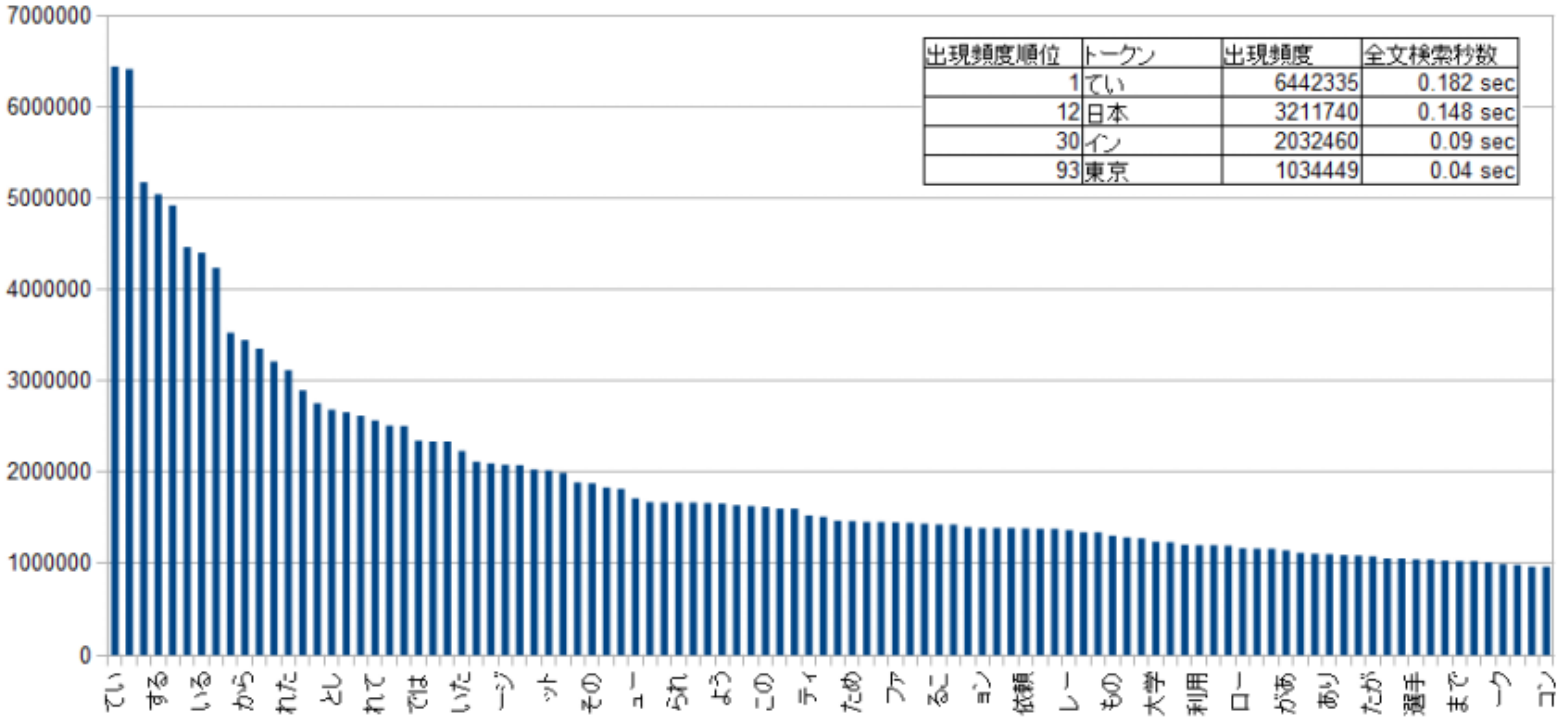
トークナイザー	キーの数	キーサイズ
TokenBigram	5767474	136.047MiB
TokenTrigram	28691883	684.047MiB

TokenTrigramのデメリット

- TokenTrigramはTokenBigramに比べキー数とキーサイズが増大
- メモリ使用量が増大
  - キーサイズは小さいほうが望ましい

Bigramトークナイザーの出現頻度と検索速度例(再褐)

Wikipedia(ja)の TokenBigram によるトークンの出現頻度上位 100 位 (全体 2,382,473)



出現頻度順位	トークン	出現頻度	全文検索秒数
1	てい	6442335	0.182 sec
12	日本	3211740	0.148 sec
30	イン	2032460	0.09 sec
93	東京	1034449	0.04 sec

Ngramトークナイザーを効率化するために

- トークンの出現頻度は大きく偏っている
- 大半のトークンは出現頻度が高くなく十分な検索速度が得られている
- Bigramの出現頻度が高い部分さえTrigramにできれば良い

Ngramトークナイザーを効率化するために

- これを追加実装したのが以下のトークナイザー
  - TokenYaVgram
    TokenYaVgramBoth
    ~SplitSymbolAlphaもあり

TokenYaVgram

- 管理テーブルのキーと一致するBigramトークンのみを後ろに伸ばしてTrigramにする
- 管理テーブルにあらかじめ出現頻度に応じたBigramトークンを登録しておく

TokenYaVgram

- 「画像処理装置」で「処理」を管理テーブルに登録
- 「画像/増処/処理装/理装/装置/置」

TokenYaVgram

- 検索クエリの末尾では、Trigram対象のBigramトークンであっても後ろに伸ばせない
- この場合は強制的に前方一致検索させる
- 「画像処理」で「処理」を登録
  ⇒「画像/　　/処理*」

TokenYaVgram/ TokenBigramの速度

Wikipedia(ja)で1000回検索

トークナイザー	検索秒数平均
TokenBigram	0.0444 sec
TokenYaVgram	0.0166 sec

TokenYaVgram/TokenBigramのキー

Wikipedia(ja)

トークナイザー	キーの数	キーサイズ
TokenBigram	5767474	136.047MiB
TokenYaVgram	7425198	172.047MiB

TokenYaVgramの効果

- キーサイズの増大を抑えつつ、検索速度の高速化を実現
- しかし、検索クエリ末尾のものは後ろに伸ばすことができない
- 「画像処理装置」で「装置」を登録
  ⇒「画像/増処/処理/理装/装置/置」

TokenYaVgramBoth

- 管理テーブルのキーと一致するBigramトークンのみを後ろに伸ばしてTrigramにする
- 1つ後ろのBigramトークンが管理テーブルのキーと一致するトークンも後ろに伸ばしてTrigramにする

TokenYaVgramBoth

- 管理テーブルにはあらかじめ出現頻度に応じたBigramトークンを登録
- 「画像処理装置」で「処理」を登録
  ⇒「画像/増処理/処理装/理装/装置/置」

TokenYaVgramBoth

- この場合、検索クエリでは伸ばせないケースが非常に多く発生する
- 全ての場合で強制的に前方一致検索させる を得ない
- 「画像処」で「処理」を登録
  ⇒「画像/増処*」

TokenYaVgramBoth/TokenBigramの速度

Wikipedia(ja)で1000回検索

トークナイザー	検索秒数平均
TokenBigram	0.0444 sec
TokenYaVgramBoth	0.0065 sec

TokenYaVgramBoth/TokenBigramのキー

Wikipedia(ja)

トークナイザー	キーの数	キーサイズ
TokenBigram	5767474	136.047MiB
TokenYaVgramBoth	8560779	200.047MiB

TokenYaVgramBothの効果

- 出現頻度が高いもののみTrigramにすることでキーサイズの増大を抑えつつ、検索速度の高速化を実現
- TokenYaTrigram並の検索速度ながらもキーサイズをTokenTrigramの1/3以下に抑えられた

既知フレーズのグループ化

- あらかじめ既知のフレーズを管理テーブルに登録
- そのフレーズのみグループ化してトークナイズ
  - パトリシアトライのLCPサーチを利用して高速にフレーズ抽出

既知フレーズのグループ化

- 「１２月は寒い」で「１２月」を登録
  ⇒「１２月／は寒／寒い」

既知フレーズのグループ化の効果

- 検索ノイズの低減
- 見出しタグや頻出語を含む複合語などを登録することにより頻出トークン数を低減

まとめ

- ✓ 検索速度を高速に保つのためにはポスティングリストが長くなりすぎないようにする
- ✓ Ngramの検索時はオーバラップさせなくても良い
- ✓ トークンの出現頻度は偏る
- ✓ これらの特性から検索を高速化、効率化するためにNgramトークナイザーを改良

おまけ その他のプラグイン

- groonga-token-filter-yatof
  https://github.com/naoa/groonga-token-filter-yatof
  - 適当なトークンフィルター集
  - LengthとかSymbolとかSynonymとか
  - 使おうとしている

おまけ その他のプラグイン

- groonga-command-token-count
  https://github.com/naoa/groonga-command-token-count
  - ポスティングリストをたどってトークン数を数える
  - 別にコマンドプラグインである必要はなかった
  - 使っている

おまけ その他のプラグイン

- groonga-function-snippet_tritonn
  https://github.com/naoa/groonga-function-snippet_tritonn
  - フルスペックスニペット関数 Mroonga(Tritonn) Like
  - 地獄のシンタックス
  - 使っている

おまけ その他のプラグイン

- groonga-tokenizer-tinysegmenter
  https://github.com/naoa/groonga-tokenizer-tinysegmenter
  - TinySegmenterを使った形態素解析トークナイザー 態素解析用の辞書を持たないのでコンパクト
  - 学習ツール公開している人がいるのでそれ使えば簡単に学習できる
  - 使っていない

おまけ その他のプラグイン

- groonga-tokenizer-yadelimit
  https://github.com/naoa/groonga-tokenizer-yadelimit
  - TokenDelimitのバリエーション
  - 使おうとしている

おまけ その他のプラグイン

- groonga-function-regex
  https://github.com/naoa/groonga-function-regex
  - RE2ライブラリを使って正規表現でoutputを整形
  - Onigumoバンドルされたからそれ使えばいい気がする
  - 使っていない

おまけ その他のプラグイン

- groonga-normalizer-yamysql
  https://github.com/naoa/groonga-normalizer-yamysql
  - ハイフンとか漢字の異体字とかヴァとかを正規化
  - フレーズ除去とか
  - 盛大にバグっている なおったら使う予定

おまけ その他のプラグイン

- groonga-column-hole
  https://github.com/naoa/groonga-column-hole
  - カラムにデータをいれたらデータが消えるかも
  - 転置索引はつくられる
  - hook apiがあることを知ったので試しただけ使っていない

おまけ その他のプラグイン

- groonga-word2vec
  https://github.com/naoa/groonga-word2vec
  - コピペしただけGroongaのプラグインである意味はない
  - 自動的にクエリ展開とかあるかも
  - 使っていない