D-032

Centroid Path Decomposition によるダブル配列の検索の高速化
A Fast Retrieval Method of Double Array Structures by Centroid Path Decomposition

上野　祐聖† 神田　峻介† 泓田　正雄† 森田　和宏† 青江　順一†
Yusei Ueno　Shunsuke Kanda　Masao Fuketa　Kazuhiro Morita　Jun-ichi Aoe

1. はじめに

単語（キー）集合を効率的に計算機により管理する方法として，トライ [1]が存在する．トライは単語の接頭辞を共有した木構造により，理論的な検索時間を単語の長さで実現できる．そのため，情報検索における索引などに用いられている． このトライを実現するデータ構造の一つとして，検索の高速性とコンパクト性を実現するダブル配列 [2][3]が存在する． また，トライにおいて，多くの単語で共有されている節を優先的に辿ることで，高速に検索する手法として Centroid Path Decomposition（CPD）[4]が存在する．本稿では，CPD をダブル配列により実現することでダブル配列の検索を高速化する手法を提案する．

2. 関連研究

2.1 ダブル配列

ダブル配列は，各節に対応する BASE， CHECK の 2 つの 1 次元配列と，単語を決定づける節（SP 節）以降の，遷移文字列（SP ストリング）を格納する TAIL から構成される．単語集合 K = {“apple”, “app”, “ape”, “baby”, “base”}に対するダブル配列を図 1 に示す． このとき，“apple”に対応する節において SP 節は 9 であり，SP ストリングは “e#”である．また，‘#’は単語の終端を表す．

節 s から t への遷移を文字 c によりおこなうとき，次の 2 式で遷移を実現する．ここで，CODE には文字 c に対応する数値が格納されている．

$$t = \mathrm{BASE}[s] + \mathrm{CODE}[c], \mathrm{CHECK}[t] = c \qquad \text{（式 1）}$$

節 s からの文字 c による遷移先 t は，BASE[s]と CODE[c]の和によって決定される．このとき，遷移先 t が正しい遷移であることを，CHECK[t]と c の一致により確認する．また，葉を除く BASE の値において重複を許さないことで，遷移文字による遷移を一意に実現する．そして，SP 節では BASE の負の値により，TAIL へのリンクを実現する．

2.2 Centroid Path Decomposition（CPD）

トライの各節において，自身の子の中で最も子孫の葉の多い節を Heavy Child（HC）と呼び，各節から HC への Path を Heavy Path（HP）と呼ぶ．CPD では HP を優先的に辿ることで，高速な検索を実現する．図 1 のトライの CPD を図 2 に示す．図の太線は HP を表している．

3. CPD の適用によるダブル配列の検索の高速化

本手法では，CPD によるダブル配列の検索の高速化をおこなう．CPD を適用するために，従来のダブル配列で用いられる BASE，CHECK，TAIL に加え，新たに一次元配列 LABEL，NEXT を用意する．

単語集合 K に対する本手法を図 3 に示す．HP 上の節は BASE，LABEL の各要素に対応する． LABEL には，HP 上の遷移文字を連続で格納する．例えば，HP 上の “apple”は

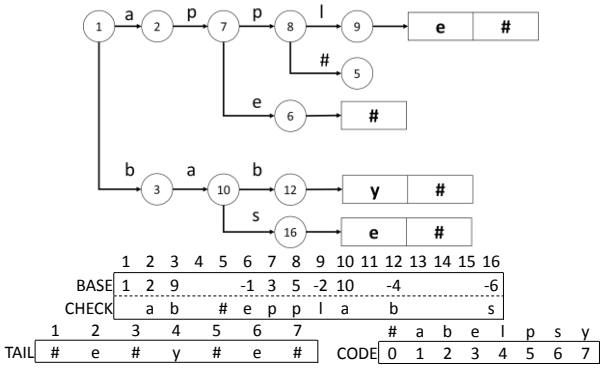

図 1 ダブル配列

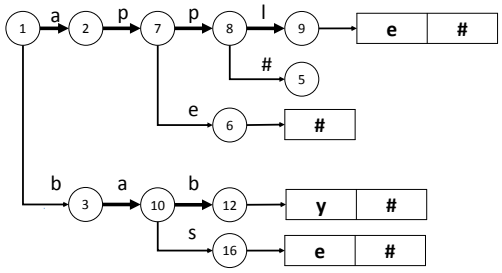

図 2 図 1 のトライの CPD

LABEL[1…5]に格納する．これにより文字列比較のみで，HP 上を高速に遷移することができる．また，BASE には HP 上にない節への遷移を実現する整数を格納する．ここで，HP 上にない節を接続節と定義する．図 3 において接続節番号は下線付き数字で表している．NEXT，CHECK の各要素は接続節に対応する．CHECK には接続節への遷移文字を格納し，NEXT には次の HP の先頭の節番号を格納する． 例えば，接続節 4 には遷移文字‘b’と次の HP の先頭の節番号 6 が対応する．

本手法によるトライの節の探索を，例を交えて説明する．本手法では HP を優先的に辿るため，クエリを LABEL の先頭から文字列比較する．例えば，HP 上の “apple”を検索する場合，節 1 から 5 を LABEL[1…5]との文字列比較のみで辿ることができる．もし節 s において文字列比較に失敗した場合，接続節 t への遷移を節 s に対応する BASE と，比較に失敗したクエリの文字 c を用いて，式 1 により O(1)で実現する．例えば，根から節 1 への HP 上の文字‘a’との比較に失敗し，接続節へ遷移文字‘b’により遷移する場合，BASE[1]と遷移文字‘b’を使って，BASE[1]+CODE[‘b’]=4 より接続節 4 へ遷移する．このとき‘b’=CHECK[4]なので正し

† 徳島大学　Tokushima University

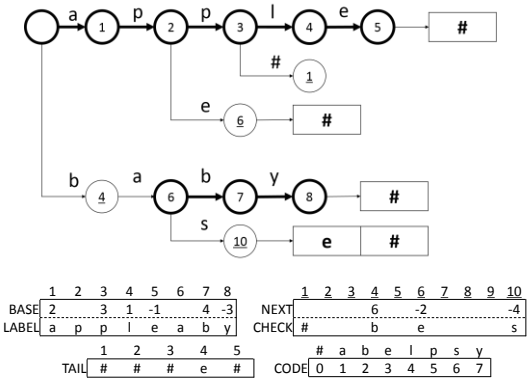

図 3 提案手法

い遷移であることが確認できる．もし CHECK が一致しなかった場合，クエリが単語集合に存在しないとして，探索を打ち切る．接続節 t から次の HP の先頭節，つまり t の HC への遷移は，t に対応する NEXT により決定する．例えば，接続節 4 において，NEXT[4]=6 より次の HP の先頭節 6 へ遷移する．

以上の探索により SP 節へ到達したとき，SP 節に対応する BASE，または NEXT の負の値により TAIL へのリンクを実現し，以降の探索は SP ストリングとクエリの未探索の文字列を比較することで達成する．例えば，接続節 10 において，NEXT[10]=-4 より TAIL[4]から“e#”を 参照する．本手法では，ある節の HP に対応する文字列との比較に失敗したとき，節に対応する BASE を用いて接続節への遷移をおこなう．そのため，トライの単語を決定づける節を，SP 節として用いることができない節が存在する．図 2 の節 9,12 は SP 節だが，図 3 の節 4,7 は SP 節ではない．そこで，従来の SP 節を利用できない節に対し，自身の子を SP 節として実現した．図 3 の節 5, 8 がこれにあたる．

本手法によって，図 3 の節 1 から 5 の遷移と 6 から 8 の遷移では式 1 を必要とせず，文字列の比較のみで辿ることができるため，高速である．また，接続節への遷移も式 1 により O(1)でおこなうため，高速である．

4. 評価

ダブル配列へ CPD を適用することで，ダブル配列の検索が高速化されたことを，実験により検証する．単語集合は日本語，英語の Wikipedia のページタイトル[1]から，ランダムに抽出した 10 万語から 50 万語とした．それぞれ単語集合に含まれる単語すべてを検索し，1 単語あたりの検索時間により，検証をおこなった．実験環境は CPU が Core i7 の 3.1GHz，L1，L2 キャッシュが 256KB，1MB，メモリが 8GB でおこなった．

図 4 に単語数と検索時間の関係を示し，図 5, 6 に単語数とデータ容量の関係を示す． Wikipedia のページタイトルの日本語を JP とし，英語を EN とする．平均データ容量は JP，EN で，提案手法が従来の 13%増となったものの，平均検索時間において，JP では従来の 91%に短縮し，EN では 85%に短縮することができた．データ容量増加の原因は，一部の SP 節をトライの後方へ移動したためと考えられる．これらの実験結果により，提案手法を適用することでデータ容量の増加はあるものの，高速な検索を実現できることがわかる．

[1] Wikipedia ページタイトル http://dumps.wikimedia.org/

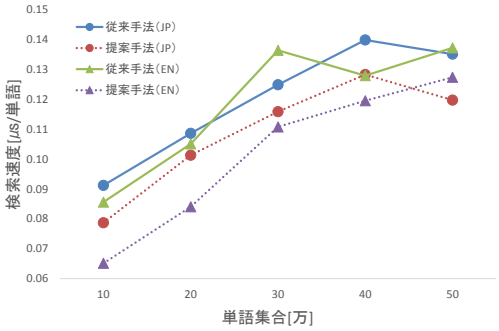

図 4 Wikipedia 日英の検索時間

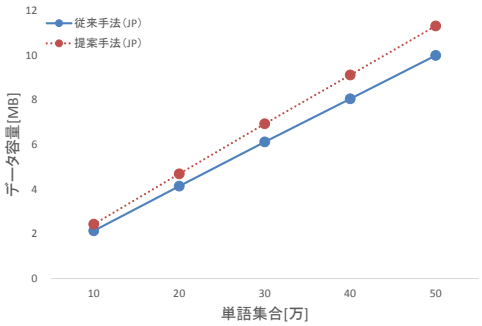

図 5 Wikipedia 日本語のデータ容量

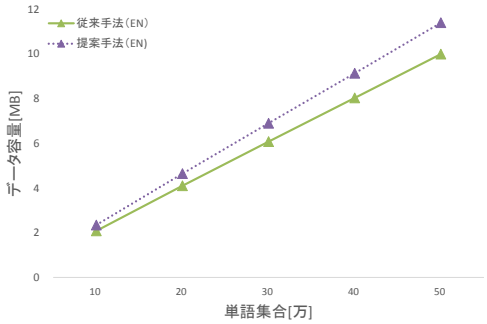

図 6 Wikipedia 英語のデータ容量

5. おわりに

本稿では，トライの検索の高速化を実現する CPD をダブル配列に適用することにより，HP 上を文字列比較のみで辿ることで，ダブル配列の検索を高速化できることを示した．今後の課題として，増加したデータ容量を削減することが挙げられる．

参考文献

[1] 青江 順一, “トライとその応用(＜連載講座＞キー検索技法 4)” , 情報処理, Vol.34, No.2, pp.224-251 (1993)
[2] 青江順一, “ダブル配列による高速ディジタル検索アルゴリズム”, 電子情報通信学会論文誌 D 情報・システム, Vol.71, No.9, pp.1592-1600 (1989)
[3] 矢田晋, 森田和宏, 泓田正雄, 平石亘, 青江順一, “ダブル配列におけるキャッシュの効率化”, FIT2006, pp1066-1077 (2006)
[4] Paolo Ferragina, Roberto Grossi, Ankur Gupta, Rahul Shah, Jeffrey Scott Vitter, “On Searching Compressed String Collections Cache-Obliviously”, PODS, pp.181-190 (2008)