6N-07

トライ木における節点の深さに基づいたダブル配列構造の構築法

Construction Method for a Double-Array Structure Based on the depth of a Node on Trie Tree

村山 智也 †
Tomoya Murayama

望月 久稔 †
Hisatoshi Mochizuki

1. はじめに

検索を目的としたデータ構造であるトライ木の実装法として，ダブル配列がある [1]．ダブル配列は領域を削減するために，トライ木における節点の遷移情報を組み合わせて 1 次元の配列に保存する．そのため，遷移情報の組み合わせ方により，同じキー集合を登録しても配列上の節点配置は異なる．そこで，節点の配置と探索性能の関係に着目して，構築済みダブル配列構造を変換することで探索を高速化する手法がある [2]．

本稿は，構築の過程で節点の深さと分岐度を指標として，ダブル配列における使用頻度の高い節点の配置を変更することで，探索が高速なダブル配列構造の構築法を提案する．

2. ダブル配列の節点配置と探索性能

ダブル配列はトライ木における節点の遷移情報を 2 本の 1 次元配列 $Base$ と $Check$ により表現する．遷移元節点 r から遷移種 a による遷移先節点 t への遷移が存在するとき，式 (1)，式 (2) が成り立つ [1]．

$$Base[r] + a = t \quad (1)$$

$$Check[t] = r \quad (2)$$

ダブル配列は定義式に基づいて 1 次元の配列に節点の遷移情報を組み合わせて保存する．そのため，同じキー集合を登録しても，節点の配置順や，遷移を追加した際に衝突した節点の再配置方法により，ダブル配列における節点の配置が異なり得る．

例として，キー集合 K ={"aa", "ac", "c"} を登録したトライ木に対応する 2 通りのダブル配列を図 1 に示す．図 1 では遷移種の内部表現値を { a = 1, b = 2, c = 3} とし，トライ木における節点の数字はダブル配列上のインデクスに対応する．左右でトライ木の形状は同じだが，ダブル配列における節点の配置は異なる．

節点の配置が異なれば探索性能が同じとは限らないと考えられ，探索に適した節点の配置による探索性能の向上が期待される．そこで，構築済みのダブル配列構造を対象として，探索がより効率的となる節点配置に変換する先行研究 [2]（以降，変換手法）がある．トライ木は登録キー集合の共通接頭辞を併合した構造をとるため，トライ木において深さの浅い節点ほど分岐が多くなりやすい．そして，複数のキーを探索する際は，根節点から葉節点に向けて遷移を繰り返すため，トライ木の浅い部分は探索において使用頻度が高いと考えられる．そこで変換手法は，分岐の多い節点を使用頻度の高い節点として，これをダブル配列上で集めた配置に変換することで探索の高速化を図る．

†大阪教育大学, Osaka Kyoiku University

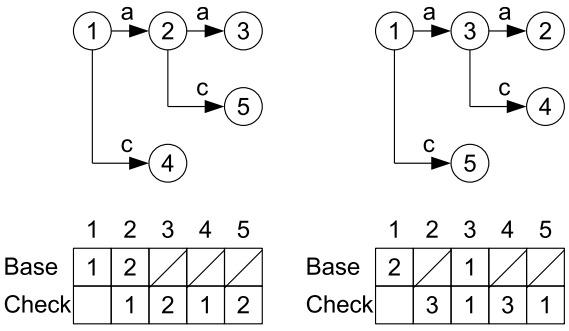

図 1: キー集合 K に対する異なる節点配置

3. 節点の深さと分岐度に基づくダブル配列構築法

トライ木に登録したキー数が少なければ接頭辞が重複しにくく，キーを登録する過程ではトライ木上で深さの浅い節点から追加されやすい．よって，浅い節点の分岐度は高くなりやすい．また，ある節点は少なくとも自身の分岐度に等しい個数の葉節点を子孫に持つ．例えば図 1 のトライ木における根節点の分岐度は 2 で，以降に葉節点が 3 個つながる．このように，分岐度が高い節点はそれだけ多くのキーが以降につながりやすく，探索で使われやすい．すなわち，浅い節点は探索での使用頻度が高いと考えられる．

そこで本稿は，トライ木における節点の深さと分岐度を指標として，構築過程で深さが浅い節点の分岐度が高くなった際，節点を効率的な配置に変更する手法を提案する．ここで，深さが d である節点の構築過程における平均分岐度を $b(d)$，構築終了時における平均分岐度を $B(d)$ とする．構築の過程で $B(d)$ は未知であるため，既存の情報を利用する．倍率 β は 0 から 1 の値をとり，値が大きければ節点配置を変更するタイミングは遅くなり，小さければ早くなる．構築過程で深さが浅い節点の分岐度を評価する式を式 (3) に示す．

$$b(d) < \beta \times B(d) \quad (3)$$

構築過程のダブル配列が式 (3) を満たさなくなった場合，提案手法はトライ木における深さが浅い節点の分岐度が高いと見なし，その時点で節点の配置を変更する．この際，分岐度の高い節点をダブル配列上で集めた配置に変更する．提案手法は節点の配置を構築過程で 1 度に限り変更する．

4. 評価

提案手法を，重越らの手法 [3]（以降，従来手法），変換手法，奈良先端大の Darts [4]，Google の darts-clone [5] と比較する．実験では，Wikipedia [6] の英語タイトルからランダムに抽出した 5 万〜50 万のキー集合を使用する．キー集合を昇順に登録したダブル配列構造に対して，20 万キーをランダムに探索する．これを 100 回繰り返した平均探索時間を評価する．

実験環境として Intel Core i7 920 2.67GHz を使用する．提案手法はトライ木において深さが浅く，分岐が多い節点を重視するため，d は実験的に 0 とする．$B(0)$ は実測値に基づき，27 に設定する．β は { 0.1, 0.2, ... 0.8, 0.9 } の中で探索時間を最も短縮した 0.9 とする．

提案手法と従来手法，変換手法の探索時間を図 2 に示す．提案手法は従来手法に比べて，探索時間を 2〜3 %短縮した．両手法の構築アルゴリズムは異なるため，出来上がるダブル配列の節点配置も異なる．ここで，両手法の探索アルゴリズムは同じであり，探索性能の差はダブル配列上の節点配置の違いにより生じたと考えられる．したがって，提案手法は従来手法に比べて，探索が効率的なダブル配列を構築したと言える．

続いて，提案手法と変換手法を比較する．提案手法は変換手法に比べて，20 万件のみ 5 %，それ以外の件数では 0〜0.7 %探索時間が長くなった．提案手法の対象が構築過程のダブル配列であるのに対して，変換手法は構築済みのダブル配列を対象とするため，トライ木における節点の深さや分岐度などの情報は変換手法の方が多く得られる．そのため，変換手法の方がより効率的な構造を構築し得る．一方，提案手法は深さが浅い節点の分岐度が高くなった際に節点の配置を変更するため，頻繁に使用する節点の配置を変更できる．よって，変換手法と近い探索性能を発揮したと考えられる．以上より，ダブル配列はその構築過程で節点の配置を変更しても，探索を充分に高速化できることが分かる．

また，提案手法が使用する d と β は登録するキー集合により変更する必要があると考えられる．今回の実験はキー集合を昇順に登録したため，トライ木を前順走査する要領で節点が追加される．よって，必ずしもトライ木における深さの浅い節点から追加されるとは限らない．キー集合をランダム順に登録する場合は，追加するキーと登録済みキーとの接頭辞が重複しにくく，浅い節点から追加されやすい．そのため，節点の追加されやすい深さに応じた d と β を指定する必要がある．

提案手法と Darts，darts-clone の平均探索時間を図 3 に示す．提案手法は Darts に比べて 11〜27 %，darts-clone に比べて 18〜34 %の探索時間を抑制した．よって，提案手法で構築した辞書を Darts や darts-clone に導入することで，探索性能が向上する可能性がある．

5. おわりに

本稿は，節点の深さと分岐度を指標として，ダブル配列の構築過程で節点の配置を変更する手法を提案した．提案手法は節点配置に着目しない従来手法に比べて探索時間を短縮した．これにより，節点の配置を考慮したダブル配列を構築することで，探索性能を向上

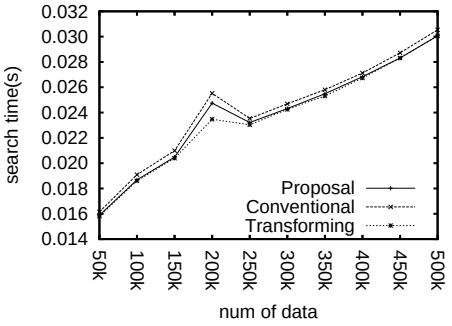

図 2: 提案手法と従来手法，変換手法の平均探索時間

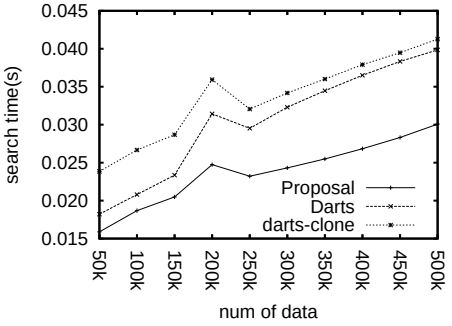

図 3: 提案手法と Darts，darts-clone の平均探索時間

できることを示した．また，構築済みダブル配列を対象とした変換手法との比較から，構築過程で節点の配置を変更しても探索性能を向上できることを示した．

今後の課題として，提案手法で構築した辞書を導入した Darts や darts-clone の探索性能の評価，提案手法の構築における d と $B(d)$ の設定方法，および倍率 β による探索性能への効果の評価が挙げられる．

参考文献

[1] Jun-ichi Aoe：An Efficient Digital Search Algorithm by Using a Double-Array Structure，IEEE Transactions on Software Engineering，Vol.15，No.9，pp.1066-1077，1989.

[2] 村山智也，望月久稔：遷移先節点に着目したダブル配列構造による探索の高速化の提案，FIT2014，第 2 分冊，pp.1-6，2014.

[3] 重越秀美，蔵満琢麻，望月久稔：ダブル配列の遷移集合管理による追加・削除処理の高速化，FIT2009，第 2 分冊，pp.1-6，2009.

[4] Darts,http://chasen.org/˜taku/software/darts/，2015.

[5] darts-clone,http://code.google.com/p/darts-clone/，2015.

[6] wikipedia,https://www.wikipedia.org/,2015.