社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ＦＩ－86 (6)
2007－ＤＤ－60 (6)
2007／3／27

ダブル配列上の遷移数を抑制した基数探索法の提案

中村 康正 †, 望月 久稔 †

† 大阪教育大学教育学部

木構造で表現される基数探索法は，共通接頭辞探索が容易であるため，自然言語処理などを中心に広く用いられている．基数探索法の探索処理を効率化するため，木構造において遷移が一つしかない分岐を圧縮したパトリシアや，木構造を多分木としたマルチウェイ基数探索法が提案されている．また，マルチウェイ基数探索法のデータ構造として，高速性とコンパクト性をあわせもつダブル配列がある．本論文では，探索速度を効率化するため，ダブル配列上の遷移数を抑制した基数探索法を提案する．評価実験の結果，提案手法が有効であるとわかった．

Proposal of Digital Search Method Reduced Search Length on the Double-Array

Yasumasa NAKAMURA † Hisatoshi MOCHIZUKI †

† Faculty of Education, Osaka Kyoiku University

Radix search method is used widely, such as dictionary information construction of the natural language processing system. Patricia and multiway radix search method is proposed in order to accelerate search processing. The double-array structure is an efficient data structure combining fast access with compactness. In this paper, we presents radix search method reduced the average of search length. The simulation results turned out that the presented method is effective.

1 はじめに

検索技法である基数探索法は，キーのある小さな部分毎を遷移とする木構造で表現される．そのため，キー全体を比較対象とするハッシュ法や二分探索木などよりも共通接頭辞探索などの実現が容易である．よって，自然言語処理の辞書管理などを中心に広く用いられている．

基数探索法の手法として，離散探索法とマルチウェイ基数探索法がある．離散探索法はキーの各1ビットを遷移とする二分木で表現するため，遷移先がただ一つである一方向分岐が多く存在する．そこで，遷移数を抑制するパトリシア[1]がある．この手法は，離散探索法における一方向分岐を削除した木構造で表現される．

マルチウェイ基数探索法はキーの複数ビットを遷移とする多分木で表現するため，離散探索法に比べて任意のキーに対する遷移数が少ない．この手法を実現する有効なデータ構造として，高速性とコンパクト性をあわせもつダブル配列[2]がある．

ダブル配列は準静的なキー集合に対して提案された手法である．そのため近年では，動的なキー集合に対応させるため，更新処理に対する提案[3, 4]がなされている．また，使用空間を小さくするため，二分木のパトリシアをダブル配列で表現した手法[5]が提案されている．

本論文では，多分木とした場合でも一方向分岐が多いキー集合に対して探索処理を効率化するため，ダブル配列上の一方向分岐を削除した基数探索法を提案する．評価実験の結果，提案手法の探索処理は有効であるとわかった．

以下，2節で基数探索法とダブル配列を説明し，3節で遷移数を抑制した基数探索法を提案する．4節で提案手法の評価を与え，5節で本論文のまとめと今後の課題についてふれる．

2 基数探索法とダブル配列

本節では，基数探索法である離散探索法とマルチウェイ基数探索法について述べる．さらに，後者を実現するダブル配列を説明する．また，例として表記記号‘#’を終端記号としたキー集合

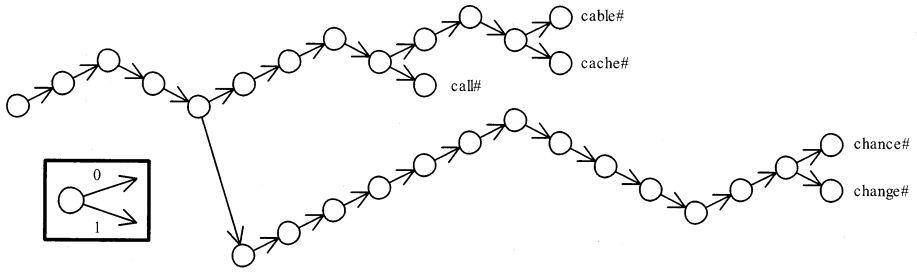


図 1　離散探索法を表現したトライ

K ={“cable#”, “cache#”, “call#”, “chance#”, “change#”}を用い，後の節でも対応させて使用する．ここで，表記記号‘#’, ‘a’, …, ‘o’の内部表現値を 0，1，…，15 とし，各内部表現値は 4 ビットで表現可能とする．

2.1　基数探索法

基数探索法のうち，木構造上の葉で，葉に対応したキーと探索キーを比較するトライがある．トライには，キー全体を木構造として実現する手法と，キーの判別に不要な遷移列を削除した最小接頭辞トライ[6]がある．後者は，前者の使用空間を抑制するため，キーが一意に判別可能な最前方の節点を葉としたものであり，トライ上で実現されていない接尾辞を葉に格納する．以下，最小接頭辞トライを単にトライとよぶ．

離散探索法は，キーの各 1 ビットを遷移とした二分木でトライを実現する．そのため，トライ上に多くの一方向分岐が存在する．そこで，トライ上の遷移数を抑制するため，一方向分岐を削除したパトリシア[1]がある．

パトリシアは，各節点にキーの比較位置を格納し，キー間で異なる遷移のみでトライを構築する．よって，離散探索法に比べて任意のキーに対するトライ上の遷移数が少ない．また，各節点に不要な遷移情報が存在せず，空間効率が良い．

パトリシアの葉には，接尾辞ではなく葉に対応したキー全体を格納する．これは，離散探索法が実現するトライとは異なり，パトリシアが実現するトライは一方向分岐を表現しないためである．よって探索処理は，トライ上を根から葉まで遷移し，葉でキーと葉が指すキー全体を比較する．

マルチウェイ基数探索法は，キーの複数ビットを遷移とした多分木でトライを実現する．そのため，離散探索法よりもトライ上の遷移数が少ない．しかし，任意の節点からすべての遷移種に対する遷移が存在するわけではないため，不要な遷移情報が多い．マルチウェイ基数探索法の探索処理は，トライ上を根から葉まで遷移し，葉で残りのキーと接尾辞を比較する．

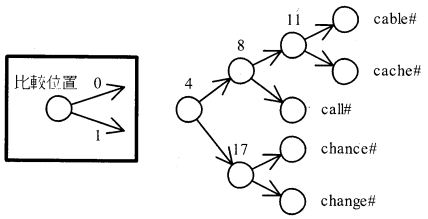


図 2　パトリシアを実現したトライ

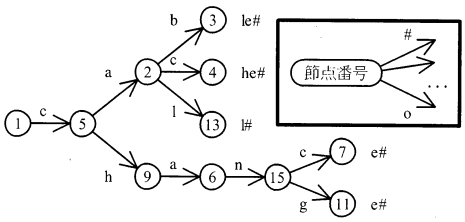


図 3　マルチウェイ基数探索法を実現したトライ

例 1: キー集合 K に対する離散探索法，パトリシア，マルチウェイ基数探索法を表現したトライをそれぞれ図 1，図 2，図 3 に示す．ここで各トライの遷移について，離散探索法とパトリシアでは内部表現値を 4 ビットの二進数で表現したものに対して 1 ビット毎とし，マルチウェイ基数探索法では表記記号毎とした．また，接尾辞をトライ上の葉の右側に示すが，離散探索法とパトリシアに対してはキー全体を示す．パトリシアについて，キーの比較位置を節点の上に示す．

以下，離散探索法，パトシリア，マルチウェイ基

数探索法について，節点数と探索処理におけるトライ上の遷移数を比較する．まず，節点数はそれぞれ 30，9，11 である．

次に，探索が成功する場合の遷移数について，“cable#”に対する遷移数はそれぞれ 12，3，3 であり，“chance#”に対しては 18，2，5 である．最後に，探索が失敗する場合について，“caching#”に対しては 12，3，3 であり，“check#”に対しては 9，2，2 である．(例終了)

使用空間について，パトリシアは離散探索法より節点数が少なく，トライ上の使用空間が小さい．また，マルチウェイ基数探索法は離散探索法より節点数は少ないが，遷移の決定を高速化するためには遷移情報を多くもつ必要がある．よって，節点毎の使用空間は離散探索法よりも大きい．

また，離散探索法とマルチウェイ基数探索法が接尾辞のみを葉に格納するのに対し，パトリシアはキー全体を格納する．よって，キー情報に関する使用空間はパトリシアが大きい．

探索処理について，二分木上の遷移数を抑制したパトリシアは離散探索法よりも遷移数が少なく，離散探索法とマルチウェイ基数探索法では多分木とした後者の遷移数が少ない．登録キー数を n，キー長を k とすると，パトリシアの探索処理は $O(\log_2 n)$，マルチウェイ基数探索法は $O(k)$ である．

2.2 ダブル配列のデータ構造

ダブル配列は，二つの一次元配列 BASE と CHECK を用いてトライ上の遷移を実現する．配列 BASE は遷移の基底位置を与え，配列 CHECK はトライ上の親子関係を決定する．つまり，図 4 に示す始点 s から遷移 a での終点 t を式 (1)，式 (2) で定義する [2]．また，トライ上の節点番号とダブル配列上の要素番号は対応している．

以下，表記記号の内部表現値を単に a とし，ダブル配列上の要素 x に関する BASE 値を $B[x]$，CHECK 値を $C[x]$ とする．

$$B[s] + a = t \qquad (1)$$

$$C[t] = s \qquad (2)$$

トライ上の葉が示す接尾辞を一次元配列 TAIL に格納する [2]．この際，葉の BASE 値を配列 TAIL の要素番号に設定する．また，他の節点と区別するために BASE 値を負値とする．以下，配列 TAIL の pos 番目の要素を $T[pos]$ とし，$T[pos]$ から始まる遷移列を $T + pos$ とする．

例 2: キー集合 K に対するダブル配列を図 5 に示す．図 5 の要素番号は図 3 の節点番号と対応している．(例終了)

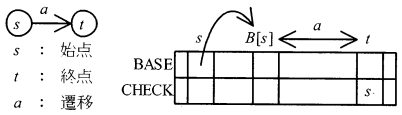


図 4 ダブル配列によるトライの遷移関係

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
BASE	2	1	-1	-4	1	1	-9		5		-11		-7		4
CHECK		5	2	2	1	9	15		15		15		2		6

	1	4	7	9	11
TAIL	le#	he#	l#	e#	e#

図 5 ダブル配列の例

ダブル配列は，離散探索法を多分木とすることでトライ上の遷移数を抑制する．しかし，図 3 における始点 1 から終点 5 や始点 9 から終点 15 のように，ダブル配列上には離散探索法と同様に一方向分岐が存在する．

3 遷移数を抑制した基数探索法

探索速度を効率化するため，ダブル配列上の一方向分岐を削除し，パトリシアを多分木に拡張する．以下，提案手法のデータ構造と探索処理について述べる．

3.1 データ構造

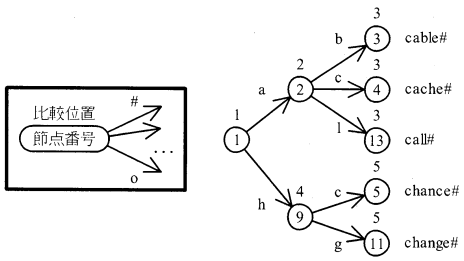
一方向分岐を削除するため，パトリシアと同様に各節点はキーの比較位置をもつ．そのため，ダブル配列にキーの比較位置を格納する一次元配列 POS を付加する．

配列 POS を用いることにより，キー key に対する始点 s からの終点 t は式 (3) となる．以下，ダブル配列上の要素 x に関する POS 値を $P[x]$ とする．また，key の pos 番目の要素を $key[pos]$ とし，$key[pos]$ から始まる遷移列を $key + pos$ とする．

$$B[s] + key[P[s]] = t \qquad (3)$$

ダブル配列とは異なり，配列 TAIL にはキー全体を格納する．これはパトリシアと同様に，根から葉まで遷移した場合でも一方向分岐を比較していないためである．

例 3: キー集合 K に対して，提案手法が実現するトライを図 6 に示す．使用空間について，提案手法が実現するトライの節点数は 8 であり，他の基数探索法よりも少ない．(例終了)



	1	2	3	4	5	6	7	8	9	10	11	12	13
BASE	1	1	-1	-7	-18				2		-25		-13
CHECK		1	2	2	9				1		9		2
POS	1	2	3	3	5				4		5		3

	1	7	13	18	25
TAIL	cable#	cache#	call#	chance#	change#

図 6　提案手法におけるトライとダブル配列の例

使用空間について，提案手法は各節点にキーの比較位置を付加したため，従来のダブル配列よりも節点毎の使用空間が大きい．またキー全体を葉に格納するため，キー情報に関する使用空間はパトリシアと同等である．

3.2　探索アルゴリズム

提案手法におけるキー key の探索が成功する条件は，式 (3) と式 (2) を満足させながらトライ上の根から葉まで遷移し，葉に対応したキーと探索キーが一致することである．

key を探索する関数 Search を以下に示す．まず，(S-1) で始点 s をトライの根に初期化する．次に (S-2) で，key に対する始点 s からの遷移を配列 POS により決定し，式 (3) により終点 t を算出する．その後 (S-3) で，式 (2) により遷移が存在するかを判断し，存在すれば (S-2) に戻りトライ上を葉まで遷移する．ここで，s が葉であるかの判断は，$B[s]$ の参照により行う．つまり，$B[s]$ が負であれば s は葉である．最後に，(S-4) で key と遷移列 $T+(-B[t])$ が一致するかを判断する．

関数：Search(key)

(S-1)：始点の初期化

始点 s をトライの根 1 に設定する．

(S-2)：トライ上の照合

もし，s が葉であれば (S-4) へ．葉でなければ，終点 t に $B[s]+key[P[s]]$ を設定する．

(S-3)：遷移の確認

$C[t]$ が s と一致する場合，s を t に更新して (S-2) へ．一致しなければ，探索失敗として FALSE を返して終了する．

(S-4)：配列 TAIL 上の照合

key と葉 t が指す遷移列 $T+(-B[t])$ を比較する．両者が一致する場合，探索成功として TRUE を返す．一致しなければ，探索失敗として FALSE を返す．

例 4: 図 6 から key = “cable#” を探索する．まず，(S-1) で始点 s を根 1 とし，(S-2) で s が葉でないことを確認する．また，$s=1$ から比較位置 $P[1]=1$ である ‘a’ で遷移する終点 t を決定する．つまり，式 (3) より t を $B[s]+a=1+1=2$ に設定する．次に，式 (2) を満足させるため，(S-3) で $C[t]=1$ が $s=1$ と等しいことを確認する．このように，式 (3) と式 (2) により，始点 1 から ‘a’ での終点が 2 であることがわかる．

同様に，式 (3)，式 (2) を確認しながら，始点 2 から $key[P[2]=2]$ である ‘b’ で終点 3 に遷移する．(S-2) で終点 3 が葉であることを確認し，(S-4) で葉 $t=3$ に対応した遷移列 $T+(-B[t])=T+1=$ “cable#” と key = “cable#” を比較する．葉に対応した遷移列と key が等しいので，探索成功として TRUE を返す．以上より，“cable#”の探索におけるトライ上の遷移数は 2 である．よって，他の基数探索法よりも提案手法の遷移数が少ない．

同様に，提案手法において “chance#”を探索した場合，トライ上の遷移数は 2 である．この場合も，提案手法の遷移数は他の基数探索法より少ない．

次に “caching#”を探索する．先程と同様に，トライ上を根 1 から 2，4 と遷移し，葉 4 に対応する遷移列 $T+(-B[4])$ = “cache#” と key = “caching#” を比較する．この場合，トライは 0 から 2 までの異なる比較位置を表現するが，両者は比較位置 4 で異なる．つまり，従来のマルチウェイ基数探索法と同様に接尾辞での探索失敗となり，FALSE を返す．また，トライ上の遷移数は 2 であり，他の基数探索法よりも少ない．

最後に “check#”を探索する．トライ上を根 1 から 9 に遷移し，式 (3) より始点 9 から $key[P[9]=4]$ である ‘k’ での終点 $B[9]+k=13$ を決定する．その後，(S-3) で $C[13]=2$ が始点 9 とは異なることがわかる．よって，始点 7 から ‘k’ での終点は存在せず，FALSE を返す．また，トライ上の遷移数は 1 であり，他の基数探索法よりも少ない．(例終了)

提案手法の探索処理は，トライ上の分岐数を b とすると $O(\log_b n)$ である．ここで，提案手法の各始点は少なくとも二つの終点をもつ．よって，b は 2 以上であり，パトリシアと比較するとトライ上の遷移数が少ない．また，多分木上の一方向分岐を

削除して遷移数を抑制した提案手法は，マルチウェイ基数探索法よりも遷移数が少ない．

4 実験による評価

提案手法の有効性を示すため，パトリシア[1]とダブル配列[2, 3, 4]を比較手法とし，探索処理，使用空間に対して比較評価を行う．

実験では，パトリシアは約400行，ダブル配列と提案手法は約700行のC言語で実装した．また，Intel Pentium4 2.8GHz，Fedora Core 4上で行い，1,000万件をランダムに抽出したURIを母集合Sとして用いた．なお，Sにおけるキーの平均長は58.47バイトであり，トライ上の各遷移をパトリシアは1ビット，ダブル配列と提案手法は1バイトとする．また，文字コードをASCIIとしたため，内部表現値は0から255である．

パトリシアについて，Morrisonの文献[1]に従って作成し，キーを配列TAILに格納した．この手法の節点は，キーの比較位置，終点への二つのリンク，キーを指す配列TAILの要素番号をもつ[1]．

探索処理について，パトリシアと提案手法は登録キー数に依存する．よって，まず登録キー数を変動させたSの部分集合S_nを用いて探索実験を行った．S_nは，Sより50万件から500万件まで50万件ずつランダムに抽出したものを用いた．また，探索したキー集合は50万件のS_nとし，すべて成功探索とした．

実験結果を表1と図7に示す．表1には，500万件のS_nに対するトライ上，配列TAIL上に対するそれぞれの探索時間，およびそれらの合計時間，トライ上の遷移数を示す．図7には，S_nに対する合計の探索時間を示す．表1と図7に示した値は，一つのキーに対する平均値である．

表1と図7より，500万件を登録した提案手法の探索処理はダブル配列より約1.93倍，パトリシアより約2.54倍高速であった．これは，表1からもわかるように，提案手法は他の手法よりもトライ上の遷移数が少なく，トライ上の探索が高速となったためである．

パトリシアに対して，提案手法は多分木であるため，各節点からの分岐数がパトリシアと比較して増加したことが遷移数の抑制につながった．500万件を登録した場合，始点からの平均分岐数は，パトリシアが2.0，提案手法が約3.0であった．

ダブル配列に対して，提案手法は一方向分岐を削除して節点数を減少したことが遷移数の抑制につながった．各手法の節点数を，後述する使用空間に関する実験結果である表2に示す．表2より，500万件を登録した場合の節点数は，約1,500万のダブル配列に対して，提案手法は約750万であり約2分の1と減少した．

表1 探索処理に対する実験結果

	探索時間（μs）			トライ上の
	トライ	配列	TAIL 合計	遷移数
ダブル配列	3.67		0.16 3.82	48.67
パトリシア	3.86		0.48 4.35	52.21
提案手法	1.52		0.46 1.98	15.97

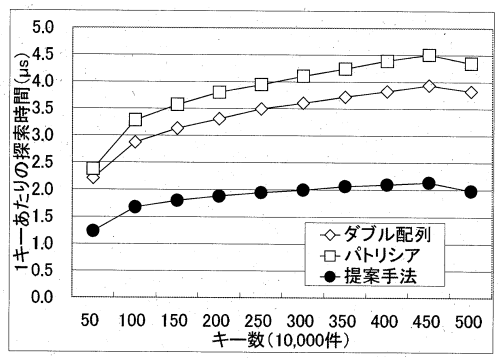


図7 登録キー数変動による探索時間の実験結果

次に，ダブル配列の探索処理がキー長に依存することから，キー長別の探索実験を行った．この実験では，Sからキー長kで切り出したキー集合S_kを用いた．ここで，kは3，...，39であり，登録キー数をそれぞれ1万件とした．

実験結果を図8に示す．図8には，S_kにおける一つのキーに対する平均探索時間を示す．

図8より，キーが23バイトよりも長くて一方向分岐を多く削除した場合，提案手法はダブル配列よりも有効であった．ここで，キー長が23バイトの場合におけるトライ上の遷移数について，ダブル配列が約10.0バイト，提案手法が約7.4バイトであり，提案手法の方が少ない．よって，提案手法がダブル配列よりも遅くなった原因として，配列POSの参照と配列TAILにおけるキー比較による負荷が考えられる．

パトリシアに対して，キー長が39バイトの場合，提案手法の探索処理は約2.15倍高速となった．これは，キー数を変動させた実験と同様に，各始点からの分岐数が増加したためである．

使用空間に対する実験について，50万，250万，500万件のS_nを用い，それぞれを登録した各手法

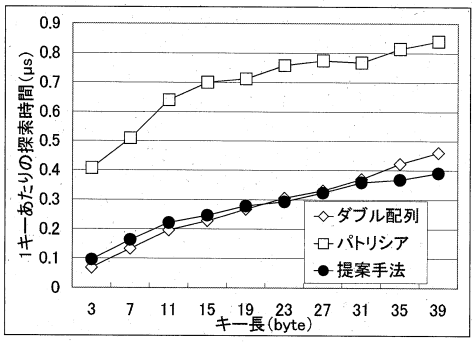


図 8　キー長変動による探索時間の実験結果

に対して，トライ，配列 TAIL，およびそれらの合計使用空間，トライ上の節点数を計測した．

使用空間に対する実験結果を表 2 に示す．表 2 より 500 万を登録した場合，提案手法におけるトライと配列 TAIL での合計使用空間は，ダブル配列の約 2.24 倍と増加し，パトリシアとは同等であった．

ダブル配列に対して，提案手法におけるトライ上の使用空間は約 2 分の 1 と小さくなった．よって，提案手法は配列 POS を付加したがトライ上の使用空間は小さくなった．一方，配列 TAIL について，キー全体を格納したことにより使用空間は約 3.98 倍と増加した．これにより，ダブル配列よりも合計使用空間が大きくなった．

パトリシア [1] について，配列 TAIL 上の使用空間は提案手法より多い．これは，キーの先頭にダミー遷移を必要とするからである．また，節点数が提案手法より少ないのは，葉とその他の節点を併用しているからである．しかし，節点毎の情報量はパトリシアの方が多く，トライ上の使用空間は提案手法と同等となった．これにより，提案手法とパトリシアの合計使用空間は同等となった．

よって実験より，探索処理および使用空間に対して，提案手法は効果的であるとわかった．

5　おわりに

本論文では，一方向分岐が多いキー集合に対して探索処理を効率化するため，ダブル配列上の遷移数を抑制した基数探索法を提案し，実験により有効性を示した．今後の課題として，半無限文字列のようなキーが長いデータに対して実験を行い，詳細に評価することが挙げられる．

表 2　使用空間に対する実験結果

キー数	50 万	250 万	500 万
節点数（10,000）			
ダブル配列	215.7	852.5	1,513.5
パトリシア	50.0	250.0	500.0
提案手法	78.1	380.9	749.5
トライ（MB）			
ダブル配列	17.26	68.20	121.08
パトリシア	8.00	40.00	80.00
提案手法	9.37	45.71	89.94
配列 TAIL（MB）			
ダブル配列	7.34	27.49	49.22
パトリシア	29.74	148.67	297.34
提案手法	29.24	146.17	292.34
合計（MB）			
ダブル配列	24.60	95.69	170.30
パトリシア	37.74	188.67	377.34
提案手法	38.60	191.89	382.28

参考文献

1) Morrison, D. R.: PATRICIA Practical algorithm to retrieve information coded in alphanumeric, *Journal of the ACM*, Vol. 15, pp. 514–534 (1968).

2) Aoe, J., Morimoto, K., Shishibori, M. and Park, K.-H.: A Trie Compaction Algorithm for a Large Set of Keys, *IEEE Trans. Knowledge and Data Engineering*, Vol. 8, No. 3, pp. 476–491 (1996).

3) 中村康正, 望月久稔：圧縮ディジタル探索木における辞書情報更新の高速化手法, 情報処理学会論文誌, Vol. 47, SIG 13（TOD 31）, pp. 16–27 (2006).

4) 矢田普, 大野将樹, 森田和宏, 泓田正雄, 吉成友子, 青江順一：接頭辞ダブル配列における空間効率を低下させないキー削除法, 情報処理学会論文誌, Vol. 47, No. 6, pp. 1894–1902 (2006).

5) 山本一徳, 獅々堀正幹, 柘植覚, 北研二：パトリシアトライの 1 次元配列構造への圧縮手法, 言語処理学会第 11 回年次大会, pp. 688–690 (2005).

6) Dundas, J. A.: Implementing Dynamic Minimal-prefix Tries, *Softw. Prac. & Exper.*, Vol. 21, No. 10, pp. 1027–1040 (1991).