Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

V-KTPY Prefix Hashed Trie for Indian Languages: A Case Study with Kannada Text Retrieval

Yashaswini Hegde, Padma S. K
Dept. of Information Science
Shree Jayachama Rajendra College of Eng.
Mysuru, India

***Abstract*— The tries used in existing retrieval engines are unicode based and are more applicable to languages like English. When same method is applied to phonetic languages like Indian and South Asian languages they aren't that efficient in terms of compact representation. Therefore, we have developed a novel V-KTPY Prefix Hashed Trie (VKTPY PHT) based approach which is efficient in memory usage (compact representation) and a general methodology for the retrieval of text in Indian and south Asian languages.**

In our approach we have made use of the fact that these languages are phonetic in nature and follows Brahmi/ Devanagari based scripts, which can be represented numerically by the "Katapayaadi Sankhya Sutra" (KTPY Rule). The enhancement proposed by us to this KTPY Rule is called "Vistruta Katapayaadi Sankhya Sutra" (V-KTPY Rule). Using Our V-KTPY Rule, we have developed an efficient encryption/decryption technique and have applied it to a south Indian language - Kannada as a case study. However, our technique is general and extendable to around 120 Indian and south Asian languages which follows Brahmi/Devanagari scripts.

Our proposed VKTPY PHT method first converts unicode Kannada to V-KTPY encoded Kannada and stores them as prefixes, indexed by its first character set. By this, we get a 20% improvement in memory compared to existing unicode tries with same time complexity.

Our proposed VKTPY PHT is not restricted to text but can also be used in speech (spoken) enabled retrieval engines for all Indian languages. We have some preliminary results to show this capability. In short, the key benefits of our approach are memory efficiency and applicability to many Indian and south Asian languages.

Keywords*— *Encryption; Compressed Trie; Kannada Prefix Hashing; NLP; Katapayaadi Sankhya Sutra ; ಕನ್ನಡ

I. INTRODUCTION

The scripts of many Indian languages like Sanskrith,Hindi, Kannada, Telagu, Marathi, Gujarathi etc. are based on Brahmi/Devanagari scrips and are similar in their structure. These languages, that are official and spoken languages of highly populated Indian states, generate huge web content. For example Kannada, a spoken language of around 60 million people from Karnataka, a southern state of India, generate millions of web pages/documents. Hence it is important to study and explore the efficient techniques for text retrieval

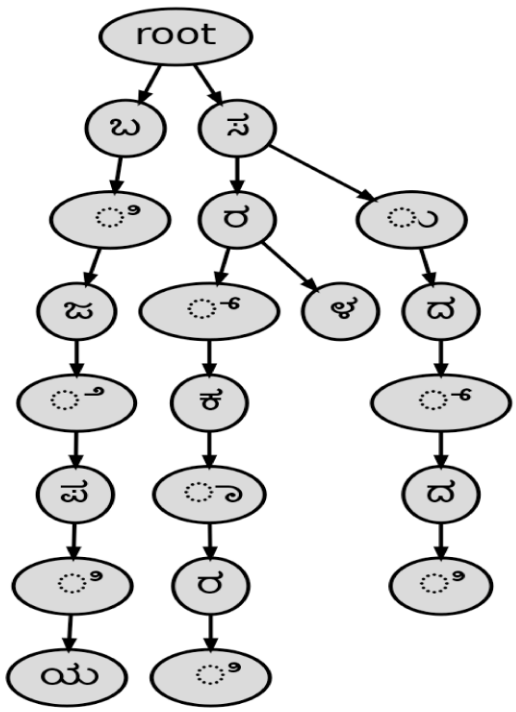

Fig.1. Simple Trie

engines (search engines) with common/similar language representations and are phonetic in nature. In this paper we are proposing a new common representation for many Indian languages and showing its efficiency in terms of space and time, when collaborated with a Trie, a basic data structure and algorithm used in text retrieval engines, taking Kannada language as a use case.

The proposed V-KTPY Rule ("Vistruta Katapayadi Sankhya Sutra") is a encryption/decryption rule by which an Indian language based on Brahmi/Devanagari script can have its numerical representation. And VKTPY PHT ("VKTPY Prefix hashed trie") is a space efficient 'prefix hashed trie' which uses VKTPY Rule to encode the input-ed unicode text.

V-KTPY Rule is based on an ancient Indian technique called The Katapayaadi Sankhya Sutra (KTPY rule). KTPY Rule is a system of numerical notation, was used by ancient

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

Indian mathematicians and grammarians as a tool to convert alphabets to numerals and vice versa.

Tries are tree based fast data structures used to store and search variable length strings but are space intensive. They are also called as digital tree or Radix tree or Prefix tree. The keys of the Trie generally are characters and all the descendants of the nodes will share the common prefixes. The frequency of the prefixes are stored in the leaves of the Trie. Tries are used to preprocess the patterns to speed up the pattern matching queries with the search time proportional to the size of the pattern. The standard simple Trie for Unicode Kannada words is given in the Fig. 1. Here each node except the root is labeled with a Kannada character. A Path from the root to the leaves gives a word. The time complexity of a Trie for insertion, deletion and search is O(dm) where d is the size of a string and m is the size of a alphabet. The space complexity is O(n) where n is the total size of strings.

The compact tries like Radix and Patricia trees are space optimized variants of Trie. The compact Prefix Tries or fully compressed Trie or the Radix Trie are obtained by collapsing the single leaf nodes. In other words the only child will be merged with its parent.

The proposed VKTPY rule which is an extension KTPY rule [7] is capable of covering around 120 [1] Indian and south east Asian languages like Balinese, Javanese etc . These languages are based on Brahmi/Devanagari and have common script structures [2].

The Katapayaadi Sankhya Sutra (ಕಟಪಯಾದಿ ಸಂಖ್ಯಾ ಸೂತ್ರ) - is a powerful encryption technique, a system of numerical notation, was used by ancient Indian mathematicians and grammarians as a tool to convert alphabets to numerals and vice versa. The oldest available evidence of the use of the KTPY rule is from ‘Grahacaaranibandhana’ by Haridatta in 683 CE. [4]

The KaTaPaYaadi technique, groups the consonants of Kannada into four. Ka, Ta, Pa and Ya, are the group names represented by the beginning letters of these groups. The rule says ‘kaadi nava’,(from ka nine letters - Velar and Palatal Stops) ‘taadi nava’ (from Ta nine letters - Retroex and dental stops), ‘paadi pancha’ (5 letters from Pa -Labial stops),‘yaadhyaShTa’ (8 letters from ya - Fricatives & Glides) assigning the values from 1 through 9 and 0 for the last letter of the group. Groups and numbers assigned for the consonants are given in Table 1. As shown in Table 1 KaTaPaYaadi can be seen as a mnemonic technique which helps to remember the numbers (i.e number to characters - decryption) and also like ASCII coding deriving numeric values to non numeric characters. [5] [3]

From Table 1 we can represent ka (k) as 1, sa(s) as 7, ma (m) as 5, na (n) as 0 and so on. So by KTPY rule the word “gaNita” (ಗಣಿತ) will become 356.

However KTPY rule has two major limitations.

- No provision for unique representations of the words.
  - A word ‘damita’ (ದಮಿತ) also becomes 356 like a word ‘gaNita’ (ಗಣಿತ)

TABLE 1. KATAPAYAADI SANKHYA SUTRA (KTPY-RULE)
ಕಟಪಯಾದಿ ಸಂಖ್ಯಾ ಸೂತ್ರ

Grp Name	1	2	3	4	5	6	7	8	9	0
Ka-grp (Velar & Palatl)	k	k^h	g	g^h	ng	c	c^h	j	j^h	ny
Ta-grp (Retroex & dental)	T	T^h	D	D^h	N	t	t^h	d	d^h	n
Pa-grp (Labial)	p	p^h	b	b^h	m					
Ya-grp (Fricatives & Glides)	y	r	l	v	sh	Sh	s	h		

- No numerical representation possible for vowels, composite and conjunctive consonants of the words.
  - Words like ‘agaNita’ (ಅಗಣಿತ), ‘uddaama’ (ಉದ್ದಾಮ) cannot be represented as numbers, as characters like ‘a’ and ‘u’ are not assigned with any representative numbers.
  - Words having compound consonants like ‘kShaNika’ (ಕ್ಷಣಿಕ}) and a word with out compound consonants like ‘kaNika’ (ಕಣಿಕ) both get same representation as 151.

These two drawbacks of the KTPY rule are addressed in the proposed “Vistruta Katapayaadi Sankhya Sutra” (ವಿಸ್ತೃತ ಕಟಪಯಾದಿ ಸಂಖ್ಯಾ ಸೂತ್ರ (V-KTPY rule)) . This V-KTPY rule

- Gives numerical representations to characters like KTPY rule.
- Gives unique representation to all characters of Kannada alphabets including ‘swara’ (vowels), ‘vargeeya vyanjana’ (classified consonants), ‘avargeeya vyanjana’ (miscellaneous consonants), ‘yogavaahaka’, ‘ottakshara’ (compound consonants) and ‘kaaguNita’ (Conjunctive consonants).
- Gives same numerical representation to alphabetical characters of many Indian and south east Asian languages that use Barhmi/Devanagari.
- Provides easy decryption rules.

The following Table 2 shows how this V-KTPY rule make use of the natural groupings of Kannada alphabets such as ‘swara’, ‘vargeeya vyanjana’, ‘avargeeya vyanjana’,‘ottakshara’ and ‘kaaguNita’- while representing them as numbers. Here each of these groups are numbered with ‘bin numbers’ from 1 to 8 and each letter belonging to these bins are numbered from 1 to 9 and 0 expect ‘pa-grp’ which is numbered from 1 to 5. This technique, thus able to cover vowels, conjunctive and compound consonants. Our proposed technique has an unique way of representing each Kannada character by a two digit number where the first digit is a ‘bin number’ and second digit is an index number of the character in that bin. The Table 2 details the V-KTPY rule. By this rule, and using Table 2 the word ‘gaNita’ (ಗಣಿತ) can be represented as 13257226, ‘Kannada’(ಕನ್ನಡ) as 1120882023, and ‘bhaashe’ (ಭಾಷೆ) as 34714781. The Table 3 gives comparative numerical representation by KTPY Rule and V-KTPY Rule.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

TABLE 2. KATAPAYAADI SANKHYA SUTRA EXTENDED (V-KTPY RULE) ವಿಸ್ತೃತ ಕಟಪಯಾದಿ ಸಂಖ್ಯಾ ಸೂತ್ರ

Grp Name	1	2	3	4	5	6	7	8	9	0
Ka-grp 1	k ಕ	k^h ಖ	g ಗ	g^h ಘ	ng ಙ	c ಚ	c^h ಛ	j ಜ	j^h ಝ	ny ಞ
Ta-grp 2	T ಟ	T^h ಠ	D ಡ	D^h ಢ	N ಣ	t ತ	t^h ಥ	d ದ	d^h ಧ	n ನ
Pa-grp 3	p ಪ	p^h ಫ	b ಬ	b^h ಭ	m ಮ					
Ya-grp 4	y ಯ	r ರ	l ಲ	v ವ	sh ಶ	Sh ಷ	s ಸ	h ಹ	L ಳ	
Swara0 5	a ಅ	Aa ಆ	i ಇ	ii ಈ	u ಉ	uu ಊ	R ಋ	Ru ೠ		
Swara1 6	e ಎ	ee ಏ	ai ಐ	o ಒ	O ಓ	ou ಔ	am ಅಂ	ah ಅಃ		
Gunita1 7	◌ಾ	◌ಿ	◌ೀ	◌ು	◌ೂ	◌ೃ	◌ೄ			
Gunita2 8	◌ೆ	◌ೇ	◌ೈ	◌ೊ	◌ೋ	◌ೌ	◌ಂ	◌ಃ		

A V-KTPY Trie is a congruence of V-KTPY rule and a Trie. It stores V-KTPY codes as labels/edges of a Trie. Fig. 1. shows a simple Trie constructed for Kannada unicode words and Fig. 2. is its equivalent V-KTPY Trie. V-KTPY codes that are stored in the Trie structure saves around 30% of the space since each letter of Kannada unicode is a multi byte character (varies from 1 to 6 bytes) and the total code size of V-KTPY encoded text will be far lesser compared to a Trie with Kannada unicode words.

VKTPY Prefixed Hashed Trie (VKTPY PHT) is a V-KTPY Trie where the V-KTPY encoded prefixes are stored as labels of the trie indexed by its first character in the node array at each internal nodes of the trie.

In VKTPY PHT, the nodes are being compressed by combining consonant conjunctives with their respective consonants because it is quite normal to treat such a combination as a single letter in Kannada lanaguage. The Fig. 3a. shows the combination of consonant conjuncts and its respective consonant in a Trie. The Fig. 3b. shows its equivalent in Kannada unicode. By comparing these two figures,it can be seen that our method further reduces the number of pointers in a Trie and thus gives more gain in using the memory space. Fig. 3c. is a fully compressed unicode Kannada tries.

TABLE 3. NUMERICAL REPRESENTAION BY KTPY RULE V/S V-KTPY RULE

Kannada Words	KTPYRule representation	V-KTPY Rule representation
ಗಣಿತ (gaNita)	356	13257226
ದಮಿತ (damita)	356	28357226
ಅಗಣಿತ (agaNita)	No representation	5113257226
ಉದ್ದಾಮ (uddaama)	No representation	552887287135
ಕ್ಷಣಿಕ (kShaNika)	151	118746257211
ಕಣಿಕ (kaNika)	151	11257211

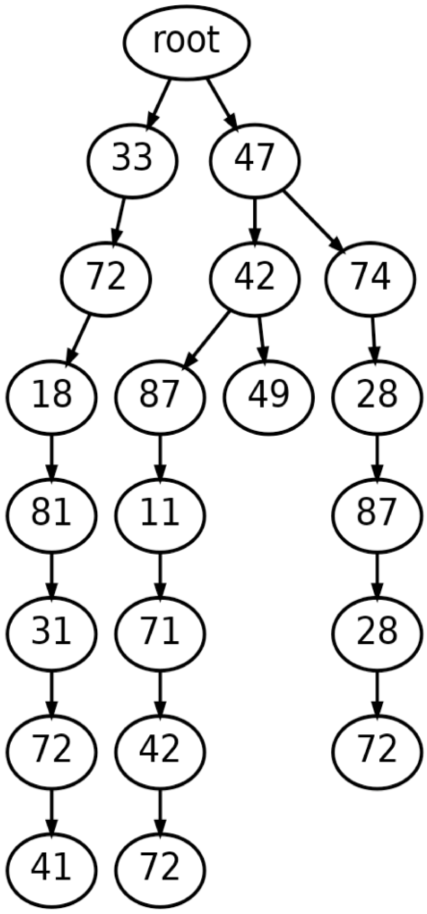

Fig. 2. Equivalent V-KTPY Trie

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

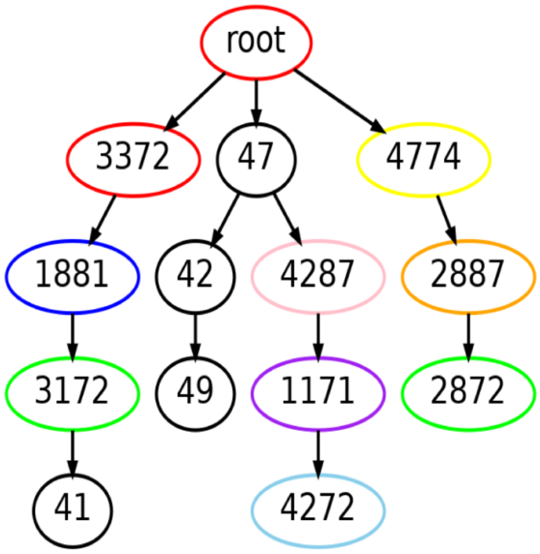

Fig. 3a. Compressed V-KTPY Trie

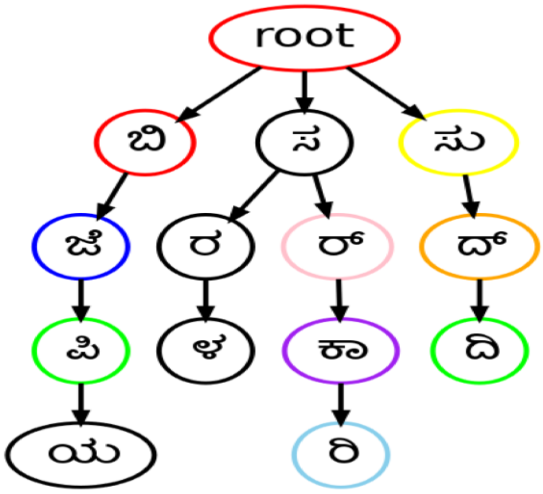

Fig. 3b. Unicoded Equivalent Trie

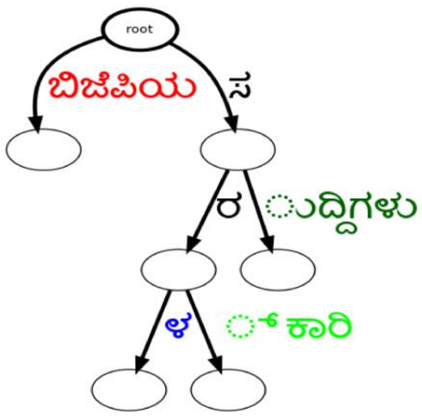

Fig.3c. Fully compressed Knnada unicode Trie

Fig.3. Compressed Tries

II. RELATED WORK

The Katapayadi sankhya sutra (KTPY Rule) though used in ancient time (1^{st} CE), it is being considered by the computer scientists recently. This technique is being considered for its powerful encoding capabilities. Anand Raman from Massey University, New zealand, compared this ancient rule, with modern hashing method [6] where till today it is believed that the concept of hashing was considered by H.P. Luhn of IBM in 1953. Subhash kak ,Regents professor of electrical and computer engineering at Oklahoma State University, observes that KTPY rules could be used like binary numbers [7] where he attributes the mapping of KTPY rules to binary values to Pingala, the brother of Panini a famous sanskrit grammarian. Further Subhash Kak along with T.R.N Rao , in his book ”Computation in Ancient India” published in 2016 [8], mentions the role of Katapayadi rule in Indian contributions to the science of computing while explaining Paninian structure of grammar and Indian logic.

Trie [14] that has been successfully used in information retrieval is an array of pointers - one for each character in an alphabet. Each leaf node is the end of a chain of trie nodes representing a string [27]. Tries are pretty fast with reasonable worst-case performance,and good for applications involving text management like pattern matching with k-d digital trie [22], dictionary and text processing with double array listing [23], MSD Radix sorting and searching [24], compression [25] and text data mining with hash-trees [20]. Tries are space-intensive, and thus makes it difficult to use with enormous volumes of strings [11]. Bell et al. in 1990 [25] shows that the space can be saved by reducing the number of trie nodes by omitting chains of tries that descend into a single leaf and thus changing their structure. They call it a compact trie [25]. In the Patricia trie Sedgewick, in 1998 [21] demonstrated that the compression is further possible by omitting all chains without branches, not just those that lead to single leaves. The ternary search tree [24] can save space by using a 3-way trie-nodes for less than, equal to, and greater than comparisons, thus reducing the node size for sparse data. Trie compression [12] , trie compaction and heuristics [26] have also been applied.

Heinz et al. in 2002 successfully reduced the number oftrie nodes at little cost, by collapsing trie-chains into buckets that share a common prefix. They called it the Burst-trie with buckets represented as linked lists with move-to-front on access as suggested by [27]. They are then selectively burst into smaller buckets that are parented by a new trie.

Generally it is perceived that Prefix Hash Trees are data structures for Distributed data bases. In 2004 Ramabhadran et al. [29] worked on Prefix Hash Tree as an indexing structure over distributed hash tables, enabling queries over distributed hash table which uses look up interfaces to construct very efficient trie based data structures.

In the context of Kannada language, tries have been used to figure out the root words by eliminating its prefixes and which are further used for the indexing purpose. Sumant Kulkarni and Srinath Srinivasa in 2013 worked on these possibilities and called this trie TrieIR [28].

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

III. VKTPY PREFIX HASHED TRIE : VKTPY PHT

VKTPY PHT is a compressed Trie having fewer branches. It is compressed by, collapsing all single nodes,by combining consonant conjunctives and also by allowing to store entire prefixes which are indexed by first characters of the prefixes.

Here each node array is a structure consisting of its prefix and a pointer to its child. Initially each word is treated as its prefix and stored in the NAry node as shown in Fig.4. Each node of the VKTPY PHT is like a hash table with each pointer, pointing to NAry tries of the subsequent levels containing V-KTPY prefix codes as hash keys. Since V-KTPY words with similar prefixes are encrypted in a similar way they can be considered as hash keys for VKTPY PHT. If a word with same prefix is hashed into the same bucket (node array), then the node is split with its prefix as a label of the root and remaining length of the key and node key (subKey1 , subkey2) as labels of its children as shown in Fig.4. For example (KannadabavuTa) is a hashed key stored as a label but indexed with its first digit in node 1. The subsequent insertion of key (kannadaasmite) hashed to the same node, causing a split in the index and also split in the node node1, into a internal node node2. The leaves leaf1 and leaf3 with its labels (baavuTa) as subKey1 and (asmite) as subKey2 are created respectively in the node node1, having (kannada) as its prefix in the node node1, which is a parent node of the node node2. The key (kannadaakshara) is also hashed to the node node1, with the same index and the prefix (kannada), navigates down the trie to reach node2. Now the words (asmite) and (akshara) are compared and the prefix (a) becomes the label of their parent node node2. The remaining letters (kShara) as subKey1 and (smite) as subKey2 will become the labels of the leaves leaf2 and leaf3 of the new node node3. After each split, the label of the parent node will get updated if its prefixKey(nodeKey) value changes due to new prefix computed. Each insertions like this begins with a search in PHT. This search gets right PHT node (addingNode) where the new key can be inserted. The following Algorithm 1 searchPHT(), explains the steps in detail. And the Algorithm 2 initInsertionPHT() and Algorithm 3 insertionPHT() explains insertion of V-KTPY Keys.

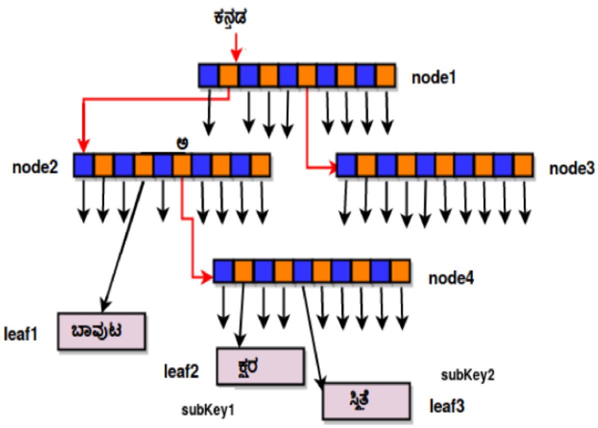

Fig. 4. VKTPY-PHT Node Structure

Algorithm1: searchPHT()
search in VKTPY PHT for suitable node to add new key

Input : ROOT, K , V -KTPY key

Initialize: Set HT← ROOT , N ← currNode, subKey1 and subKey2 are part of nodeKey and K excluding their common prefix, lpfx the length of the prefix

Get index from K

Get chains from HT.htable

Get childNode from chains

Set childNode.pfxKey as nodeKey

if nodeKey and K same **then**

    return frequency count of the K

else

    Get prefix, subKey1 ,subKey2, lpfx by calling getPrefix(nodeKey, K)

end

while children exists **do**

    Get index from subKey2

    if index in children **then**

        Get childNode from children

        Set childNode.children to children

        Set childNode.pfxKey to nodeKey

        Navigate sub tries of PHT by getting

        prefix, subKey1 ,subKey2 , lpfx on calling getPrefix(nodeKey, sunKey2)

    else

        Key Not found

        Break

    end

    return childNode,prefix,subKey1,subKey2,lpfx

end

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

Algorithm 2:initInsertionPHT()
Variable initialization part of insertionPHT()

1 Initialize: Set *HT*← *ROOT* , *N* ← currNode , nodeKey ← currentNode.PfxKey, *K* is V-KTPY key

2 insertPHT() invokes searchPHT() and K is searched in the trie, if K is not found in the trie a new node with K is inserted into the trie in the appropriate place, and returns an updated V-KTPY populated trie. If k is found then frequency count of the K is incremented.searchPHT() returns common prefix between K and nodeKey, subKey1←(nodeKey-prefix), subKey2 ←(V-KTPY Key-prefix) and length of prefix lpfx

3 *N*, keyIndex ← searchTrie(*P*,*K*);The return enumerates three cases

4 case1. *N* is root (Hash Table) when trie empty
case2. *N* is node with having Hash chain

5 case3. *N* is the node where the keyIndex not found in its chains ;

Algorithm3: insertionPHT()
adding V-KTPY keys into VKTPY PHT_

1 **if** Hash chain Empty*: case1*
then Empty Hash Table *N* is created; Get index as first character of key *K*

2 Create a new hash node hn containing *N* as its parent, index as the keyIndex of the hash chain and *K* the key

3 **return** HT

4 **if** Hash chain exists :case2 **then**

5 Check indexs present in the chain search further with searchHT() returns addingNode, prefix, subKey1, subKey2, lpfx

6 Get the children from the addingNode and prefixKey = nodeKey from the addingNode

7 **if** the nodeKey and KatapaKey are same **then**

8 Increment the frequency count of the prefixKey

9 **else if** prefix is same as nodeKey with empty sybKey2 **then**

10 Increment frequency count of the nodeKey

11 **else if** nodeKey is prefix and subKey2 is not empty **then**

12 create new hash node with addingNode as parent and prefix as its key indexed by index of subKey2.

13 Do the same even if prefix is part of the katapaKey but index the new node with index of subkey1

14 **else if** prefix is empty but subKey2 is not empty **then**

15 create a new hash node with subKey2 and addingNode as its parent indexed by first character of subKey2

16 **else if** none of the above condition satisfies and if addingNode has no children **then**

17 two hash nodes have been created one with subKey1 and other with subKey2 having addingNode as their parent indexed by their respective first characters.

18 **else if** none of the above condit ion satisfies but addingNode had children. **then**

19 This indicates that the insertion is happening in between the nodes. Then check if insertion happening between root Hash Table HT and addingNode or insertion is happening between two hash nodes.

20 **else if** insertion is is between root Hash table and a node **then**

21 insert the newly created hash node hn1 between Hash table and addingNode as its new child with prefix as its nodeKey and create another hash node hn2 with subKey2 as its nodeKey and hn1 as its parent.

22 **else**
insertion is between two hash nodes

23 repeat the above step.parent of adding node will be a hash node instead of Hash table.
End

24 else

27 index not in chains :case3

28 create a hash node with Hash table as its parent and katapaKey as its nodeKey indexed by it first character.

29 End

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 06, June-2019

To analyze these algorithms, we consider a very generic equation for M-ary tries containing N numbers of words (keys) as given by Knuth [27]. Let us say A_N be the average number of internal nodes. Then for N>=2 the equation to find the average number of nodes is

$$A_N = 1 + \sum_{k_1+\ldots+k_M=N} \left(\frac{N!}{K1!\ldots km!} M^{-N}\right)\left(A_{k_1} + \ldots A_{k_M}\right)$$

This equation can be rewritten by considering that k_1 of the keys are in the first subtrie and k_M are in M^{th} subtrie. We can rewrite the above eqation "(1)" with A0=A1=0 and if we sumup k_2 to k_M we get

$$A_N = 1 + M^{1-N} \sum_{k_1+\ldots+k_M=N} \left(\frac{N!}{K1!\ldots km!}\right) A_{k_1}$$

$$A_N = 1 + M^{1-N} \sum_{k} \binom{N}{k} (M_1)^{N-k} A_{k_1} for N \geq 2$$

If C_N is the average total number of digits inspection needed to compare all N keys in the trie then $C_0 = C_1 = 0$ and equation becomes

$$C_N = N + M^{1-N} \sum_{k} k\,(M-1)^{N-k} C_k for N \geq 2$$

Simplifying all these complicated equations we can arrive at the following description as Knuth suggests in [27] such that the number of nodes needed to store N random keys in M-ary trie with branching terminated for s keys is approximately

numNodes = N/(s \ln M) (1)

This equation "(1)" is valid for large N small s and small M and for a trie with M link field the equation further simplifis to equation "(2)"

numNodes = N/(s \ln M) if s= M (2)

IV. EXPERIMENTAL SCENARIOS AND RESULTS

Both V-KTPY encryption and VKTPY PHT have been implemented in Python. Our work uses the data set created by political articles by Shekhar Gupta's column 'RaaShtrakaaraNa' publishing in renowned daily called 'PrajavaaNi' () a Kannada news paper. There are around 44 articles containing around 6000 words. Selected stop words have been removed by own tool developed earlier [32]. Table 4 gives the details about the data set used.

TABLE 4: DETAILS OF THE DATASET

Data Set	Total words in Doc w	Number of unique words u	Number of repeated words r	Number of stop words s	Total words4 w-s = u+r
Prajavani articles	6300	2426	1465	2409	3891

The tokens of parsed documents are encoded with V-KTPY rule and then inserted in to VKTPY PHT data structure. The results are provided under different sections.

A. *Memory taken by V-KTPY encryption and Kannada unicode*

Our experiments show that Kannada encrypted using V-KTPY rule takes less space compared to Kannada unicode. In the case of the unicode representation, the character size varies from 1 byte to 6 and in V-KTPY encription method each character takes only symbolic 8 bits. The graph in Fig.5. shows that around 30% memory is saved just by using V-KTPY encrypted code, for the text corpora developed by us [10].

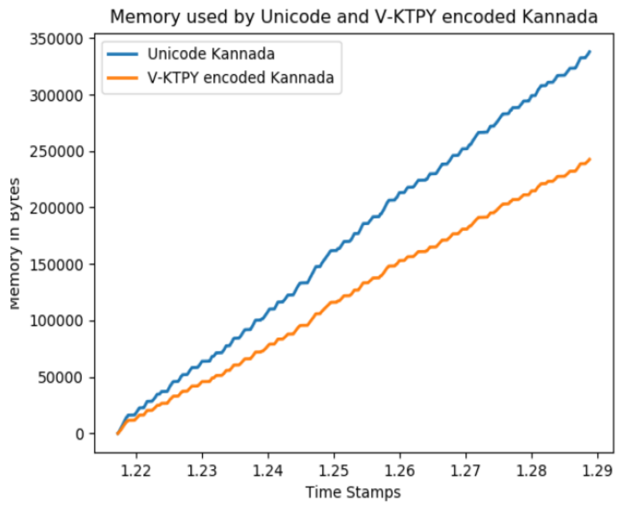

Fig. 5. Kannada Unicode word size vs V-KTPY encoded Kannada word size

B. *Memory and time taken by VKTPY PHT and fully compressed Kannada unicode Trie*

The fully Compressed Kannada unicode Trie (Fig.3c.) is compressed by converting long chains of single-child nodes to one single node. And VKTPY PHT is further compressed by merging of nodes of consonants and conjunctive consonants. When this structure is experimented with the text corpora [10], we get almost same time for inserting and searching on an average with much less space requirement. The memory used vs time ticks for inserting V-KTPY code into VKTPY PHT and insertion of Kannada unicode words into fully compressed unicode trie are shown in the Fig.6. Figure shows VKPTY PHT is more compressed as the number of words increases. Which means it is much more suitable for Big Data.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

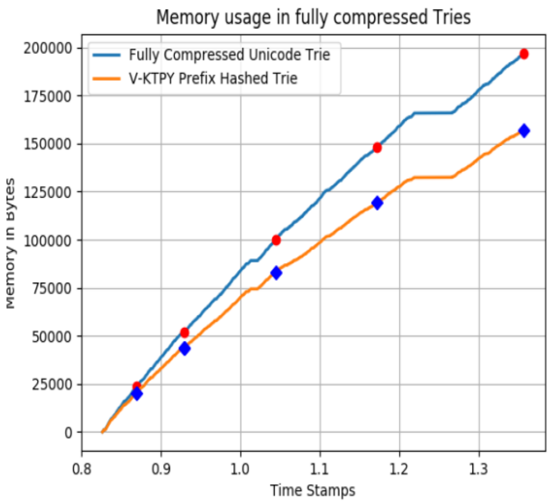

Fig.6: Insertion: Memory vs Time taken by VKTPY PHT and fully compressed Kannada unicode Trie

Table 5 gives the comparison of field links (pointers) created by fully compressed unicode trie and VKTPY PHT. From this Table 4 we can conclude that number of nodes created in case of VKTPY PHT is less and around 20% memory is saved.

Fig.7. shows the comparison of average search time taken by VKTPY PHT and a fully compressed Kannada unicode Trie.

TABLE 5: DETAILS OF THE FIELD LINKS IN TRIES

Tries	Fully Compressed unicpde Trie	Proposed VKTPY-PHT
Number of field links	197232	157304

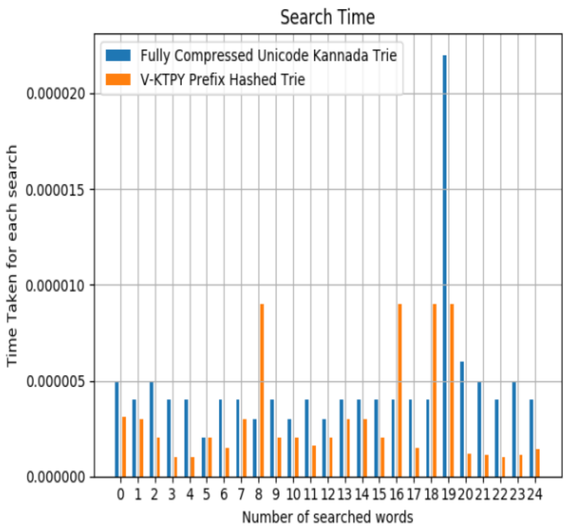

Fig. 7. Search time taken by fully compressed Unicode Kannada Trie and VKTPY PHT

The experiments with prefixed hashes shows that the encrypted code is suitable for hashing since it falls into good distribution among the nodes of the proposed data structure VKTPY PHT. Fig.8. shows the random sample of 100 words with its sorted frequencies. The graph is fitted for half Gaussian distribution. From this figure we can deduce that the spread is good and the maximum number of collisions do not exceed 20 hits for around 500 words.

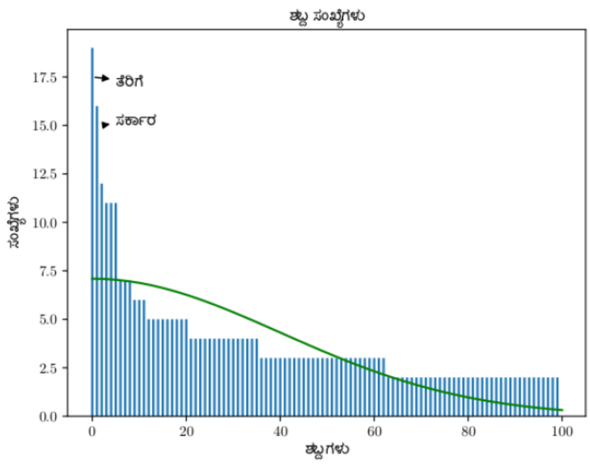

Fig.8. Curve fitted for 100 Kannada Words with sorted frequencies

V. CONCLUSIONS

Our study and experiments show that our proposed V-KTPY encryption technique is a very efficient method. It not only saves the space greater than 30% but also enables several advantages due to its capability of representing the Kannada letters numerically and symbolically. One such advantage proved in our study is effectively compressed VKTPY PHT over fully compressed Kannada unicode trie. This space and time efficient VKTPY PHT, saves the memory usage by 20% with almost the same time complexity with added possibility of phonetic hashing. With our proposed V-KTPY rules, it is easy to extend it for other Indian languages to enable cross-language retrieval. In short our study indicates that the proposed V-KTPY rule and VKTPY PHT are very efficient and effective in searching and retrieving information from Kannada documents.

REFERENCES

[1] https://scriptsource.org/scr/Dev
[2] https://en.wikipedia.org/wiki/Brahmic_scripts
[3] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.9659&rep=rep1&type=pdf
[4] https://en.wikipedia.org/wiki/Katapayadi_system
[5] https://mysteriesexplored.wordpress.com/2011/08/24/amazing-encryption-technology-in-ancient-india-the-katapayadi-shankya/
[6] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.9659&rep=rep1&type=pdf
[7] Subhash Kak, 2000, "Indian binary numbers and the Katapayadi notation", Annals of the Bhandarkar Oriental Research Institute, vol.81, 2000, pp.269-272
[8] T.R.N. Rao and Subhash kak, August 8, 2016, "Computation in Ancient India", Paperback – August 8, 2016
[9] http://www.prajavani.net/news/category/22885.htm
[10] https://drive.google.com/drive/folders/MFrrFZLNhr7F0Ge9XOLgWeTgIxL2IyX
[11] Comer, D. 1979, "Heuristics for trie index minimization", ACM trans. on Database Systems 4(3), 383–395.
[12] M. Al-Suwaiyel and Ellis Horowitz, 1984, "Algorithms for Trie Compaction", ACM Trans. Database Syst., v.9, pp.243-263
[13] Flajolet, P. & Puech, C. 1986, "Partial match retrieval of multimedia data", Jour. of the ACM 33(2), 371–407.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

[14] Fredkin, E. 1960, ”Trie memory”, Communications of the ACM 3(9), 490–499.
[15] Frigo, M., Leiserson, C. E., Prokop, H. & Ramachandran, S., 1999, ”Cache-oblivious algorithms”,in ’IEEE Symp. on the Foundations of Computer Science’, IEEE Computer Society Press, p. 285.
[16] Fu, J. W. C., Patel, J. H. & Janssens, B. L.,1992, ”Stride directed prefetching in scalar processors”, in ’Proc. Annual ACM/IEEE MICRO Int. Symp. on Microarchitecture’, IEEE Computer Society Press, pp. 102–110.
[17] Hallberg, J., Palm, T., & Brorsson, M. 2003, ”Cache-conscious allocation of pointer-based data structures revisited with hw/sw prefetching”, in ’2nd Annual Workshop on Duplicating, Deconstructing, and Debunking’.
[18] Harman, D., 1995, Overview of the second text re-trieval conference (TREC-2),in ’Proc. Second Text Retrieval Conference’, Pergamon Press, Inc., pp. 271–289.
[19] Heinz, S., Zobel, J. & Williams, H. E., 2002,”Burst tries: A fast, efficient data structure for string keys”, ACM trans. on Information Systems 20(2), 192–223.
[20] Rakesh Agrawal and Ramakrishnan Srikant, 1994 , ”Fast Algorithms for Mining Association Rules in Large Databases”, Proceeding VLDB ’94 Proceedings of the 20th International Conference on Very Large Data Bases Pages 487-499, Morgan Kaufmann Publishers Inc. SF,USA
[21] Robert Sedgewick, 1998 ,” Algorithms in C++, Parts 1–4: Fundamentals, Data Structure, Sorting, Searching”, Third Edition, Addison Wesley 1998, chapter 15.
[22] Philippe Flajolet, Claude Puech, April 1986 ”Partial match retrieval of multidimensional data”, Journal of the ACM (JACM) JACM Homepage archive Volume 33 Issue 2, Pages 371-407, ACM New York, NY, USA
[23] Aoe et al. 1992 , ”An Efficient Implementation of Trie Structures”, SOFIWARE—PRACTICE AND EXPERIENCE,VOL.22(9), 695–721 (SEPTEMBER 1992)
[24] Jon L. Bentley , Robert Sedgewick , 1997, ”Fast Fast algorithms for sorting and searching strings”, Proceeding SODA ’97 Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms , Pages 360-369, Society for Industrial and Applied Mathematics Philadelphia, PA, USA ©1997
[25] Bell, T. C., Cleary, J. G. & Witten, I. H. 1990, ”Text Compression”, Prentice-Hall. ISBN:0-13-911991-4
[26] Comer, D. 1979, ”Heuristics for trie index minimization”, ACM trans. on Database Systems 4(3), 383–395.
[27] Knuth, D. E., 1998, The Art of Computer Programming: Sorting and Searching, Vol. 3, second edn, Addison-Wesley.
[28] Sumant Kulakari,Srinath Srinivasa ,”TrieIR: Indexing and Retrieval Engine for Kannada Unicode Text”,Digital Libraries: Social Media and Community Networks: 15th International Conference on Asia-Pacific Digital Libraries, ICADL 2013, Bangalore, India, December 9-11, 2013. Proceedings (pp.21-24)
[29] Sriram Ramabhadran, Sylvia Ratnasam, ”Prefix Hash Tree An Indexing Data Structure over Distributed Hash Tables”, 2004
[30] http://ntz-develop.blogspot.in/2011/03/phonetic-algorithms.html
[31] https://thottingal.in/blog/2009/07/26/indicsoundex/
[32] Y. Hegde, S. Kadambe and P. Naduthota, ”Suffix stripping algorithm for Kannada information retrieval,” 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Mysore, 2013, pp. 527-533. doi: 10.1109/ICACCI.2013.6637227

(This work is licensed under a Creative Commons Attribution 4.0 International License.)