A Comparative Study of Data Structures for Punjabi Dictionary

G S lehal, Kulwinder Singh
Department of Computer Science and Engineering, Punjabi University, Patiala, India.

Abstract

In this paper, the implementation issues involved in an electronic dictionary for Punjabi have been studied. The commonly used data structures for English dictionary have been modified to suit the non-linear nature of Punjabi. A comparison is made of the time and space used by these data structures and conclusions made about their suitability for different applications.

Keywords : Punjabi Dictionary; Data Structures, Trie, Binary Search tree

1. Introduction

Electronic dictionaries are now being widely used for word processing, natural language processing and post processing in OCR systems. The structure of the dictionary is of great importance: it has a large effect on the performance of overall system and must therefore allow very fast searches. There are a number of ways in which a dictionary can be organised depending upon the core memory, access speed required and application area of the dictionary. A survey of adequate data structures for the representation of dictionaries is given by Knuth[1], Kukich[2] and Takahashi *et al*[3]. Peterson[4] discusses some of the organizations for interactive spelling checker application.

The structure of the Gurmukhi script, the script for Punjabi, is non-linear i.e. besides 41 consonants of the language, there are other symbols such as Laga, Lagakhar etc, which are described in detail in next section, which are used to represent the phonetic structure of the word. These symbols inherently decorate the consonant. For example, the word 'COMPUTE' as written in English, the character 'O' is called the colleague of 'C', 'M' is called the colleague of 'O' and so on, but in Punjabi it will be written as 'ਕੰਪਯੂਟ' where character ਕ is said to be wearing a cap, ੫ is holding a stick and ਊis wearing shoes. Keeping in mind this non-linear structure some of the most common data structures for electronic English dictionary have been modified to suit the needs of Punjabi dictionary. A study and comparison of these structures in terms of time and space is made and discussed in this paper.

2. Punjabi Language and Gurmukhi Script

Gurmukhi Script consists of 41 alphabets called *Vianjan.* There are 10 vowels symbols called *Laga.* The various types of symbols used to write the vowels are listed in figure 2 Of these *Kanna* and *Bihari* are written after, *Sihari* before, *Dulankarh* and Onkarh under, and *Horha, Kanorha, Laanv, Dulaanv* over the letters they vocalize. *Tippi* (ੰ) and *Bindi* (ਂ) are the two symbols to accommodate nasal sounds. *Adhak* (ੱ) is used for reduplication of sound of any consonant. They (*Tippi, Bindi, Adhak*) are called *Lagakhar* because they always come in combination with *Laga.* We also have two half characters or *Sanyukat akhar* (੍ and ੍) placed at the feet of the consonants.

It has been seen that the arrangement of letters here is more systematic than that of the English alphabets, those of each class and subclass being placed together. Thus, three vowels forms stand at the head, followed by four classes of consultant. First, we have the Sibilant and Aspirant, each of these classes being represented by one letter only; then the mutes, subdivided into five classes, each containing five letters; and finally the five semivowels; if the mutes be read in columns downwards it will be found that the five classes are arranged in order of organs by the aid of which they are pronounced, beginning with throat and ending with lips. The complete set of consonants are shown in figure 1.

Fig 1 : Punjabi Language Alphabet

Vowel	Symbol	Name
1. ਅ	-	ਮੁਕਤਾ (Mukta)
2. ਆ	ਾ	ਕੰਨਾ (Kanna)
3. ਇ	ਿ	ਸਿਹਾਰੀ (Sihari)
4. ਈ	ੀ	ਬਿਹਾਰੀ (Bihari)
5. ਉ	ੁ	ਔਂਕੜ (Onkarh)
6. ਊ	ੂ	ਦੁਲੈਂਕੜ (Dulankarh)
7. ਏ	ੇ	ਲਾਂ (Laanv)
8. ਐ	ੈ	ਦੁਲਾਂ (Dulaanv)
9. ਓ	ੋ	ਹੋੜਾ (Horha)
10. ਔ	ੌ	ਕਨੌੜਾ (Kanorha)

Figure 2 : Table showing the Laga of Punjabi Language

3. Data Structures Implemented for Punjabi

The English script has been designed in such a way that it can be used linearly. No English character ever wears a shoe, or cap, or holds a stick like the Punjabi characters. The *Gurmukhi* script is the sequence of its consonants called *Vianjan* with its associated vowel symbols called *Laga* and any other symbols such as *Lagakhar* or *Sanyukat akhar* as mentioned in previous section. So, keeping in mind the features of *Gurmukhi* script, we have implemented different data structures for Punjabi dictionary along with the different data structures for Punjabi word format. The various data structure for Punjabi dictionary are:

3.1 Binary Search Tree

The simplest data structure that was implemented and tested is the binary search tree. In each node of the tree a complete Punjabi word is stored. Ten different binary trees , where each tree stored words of same length, were simultaneously used. For determining the word length, the vowel symbols *Laga* and other symbols such as *Lagakhar* or *Sanyukat akhar, were* treated as a single character.

3.2 Trie

The idea of trie memory, which takes advantage of the redundancy of common prefixes, was first introduced by Fredkin [5]. According to Fredkin, a trie is a particular tree structure that contains only one character at each node. Each node may have ordered links to descendant nodes whose number maximally equals the number of the letters of the underlying alphabet. In particular, the root of a trie has up to n descendants where n corresponds to the first letters of a fixed vocabulary. Word access in a trie is strictly performed character by character beginning at the root of the trie. Nodes that represent the last character of a word are always marked by a special flag stopping the search process.

Tries are attractive because of their simple and compact storage allocation. Common prefixes are stored exactly once. In addition, they allow the integration of sophisticated error correction algorithm for dealing with noisy input, although problems arise when beginning of a word is erroneous. For large dictionaries the trie is not the best choice because considerable time would be spent traversing links from one character to the next. Another disadvantage is the waste of storage due to storing additional pointers and housekeeping information.

For Punjabi dictionary, we have further implemented two types of tries. The first trie is the one suggested by Sukhjeet[6] which takes care of non-linear nature of the Gurmukhi script. Each node in a trie is represented by the following structure:

```
RECORD Node
        BEGIN
                Syllable        :       SylllableType
                Flag            :       Boolean
                UpPtr           :       Pointer to Node
                DownPtr         :       Pointer to Node
        END

RECORD SyllableType
                BEGIN
                        Vianjan     :   CharacterType
                        Laga        :   VowelType
                        Lagakhar    :   LagakharType
                END
```

BITS	LAGAKHAR	BITS	LAGA
00	-	0000	Mukta
01	Bindi	0001	Kanna
10	Tippi	0010	Sihari
11	Adhak	0011	Bihari
		0100	Laanv
		0101	Dulaanv
		0110	Onkarh
		0111	Duankarh
		1000	Horha
		1001	Kanorha

Figure 3 : Bit Fields for Lagakhars and Lagas of Punjabi

In the structure *SyllableType*, the *Vianjan* field will store the main consonant and will be of character type. Since there are 41 main consonants and 2 *Sanyukat akhar* in Punjabi, so this field will require at the most 6 bits giving total of 64 bit combinations out of which first 43 will mark the different *Vianjans* and *Sanyukat akhar*.

The *Laga* field will be used to store the related vowel with consonant. Since there are 10 vowels in Punjabi so this field will require at most 4 bits giving total of 16 bit combinations out of which first 10 will mark the different vowels and rest 6 will be Don't care condition as shown in figure 3.The *Lagakhar* field will store the associated symbol for representing nasal or dual sounds. There are three such symbols in Punjabi namely *Bindi*, *Tippi*, and *Adhak*, so the field will be of length 2 bits as shown in figure 3. So, the *Syllable* will store the *Vianjan* and its associated symbols such as *Laga* and *Lagakhar*.

The field named *Flag* is a boolean field which will have value 'TRUE' if the *Syllable* part of the node is the last one in the word. This will mean that at this point the word can be considered as complete Punjabi word. The last two fields are similar as they both store the address of another node. The *UpPtr* pointer will be used to maintain the list syllable that can come at the same position in the word at the same level in the tree, whereas the *DownPtr* pointer will be used to store the list of syllable that can come after the occurrence of current syllable i.e. list of nodes at the next level of trees.

In this structure the root will contain the starting syllable for the group of words to be stored in the tree and its *DownPtr* pointer will point to the list of words that can occur at the second level from the start. Similarly *DownPtr* pointers of all the nodes at the second level of the tree will point to the list of syllables maintained by the *UpPtr* pointers, that can come at the third position from the beginning and so on.

In the second implementation of trie, each node is represented by the following structure:

```
RECORD Node
    BEGIN
        Splitchar   :   CharacterType
        Flag        :   Boolean
        UpPtr       :   Pointer to Node
        DownPtr     :   Pointer to Node
    END
```

In this structure, the *Splitchar* field is of character type which will have only one character, which may be a *Vianjan*, or a *Laga*, or a *Sanyukat akhar*. The role of *UpPtr* and *DownPtr* pointers is similar to that as described earlier in this section. The *Flag* Field is also similar to as mentioned in this section.

3.3 Ternary Search tree

Another data structure to store the dictionary is ternary search tree, which is suggested by Jon Bentley and Bob Sedgewick [7]. Ternary search trees combine attributes of binary search trees and digital search tries. Like tries, they proceed character by character. Like binary search trees, they are space efficient, though each node has three children, rather than two. A search compares the current character in the search string with the character at the node. If the search character is less, the search goes to the left child. If the search character is greater, the search goes to the right child. When the search character is equal, though, the search goes to the middle child, and proceeds to the next character in the search string. Each node in a ternary search tree is represented by the following structure:

```
RECORD Node
        BEGIN
            Syllable    :   SyllableType
            Flag        :   Boolean
            LowPtr      :   Pointer to Node
            EqPtr       :   Pointer to Node
            HighPtr     :   Pointer to Node
        END

RECORD SyllableType
        BEGIN
            Vianjan     :   CharacterType
            Laga        :   VowelType
            Lagakhar    :   LagakharType
        END
```

The *SyllableType* and the *Flag* field are similar to that as mentioned earlier. The last three fields store the address of another node, but their significance lies in different context. The *LowPtr* pointer will be used to maintain the list of syllables that are less than syllable at the current node; the *HighPtr* pointer will be used to maintain the list of syllables that are greater than syllable at that node, whereas the *EqPtr* pointer will be used to store the list of syllables that can come after the occurrence of syllable at that node.

In this structure the root will contain the starting syllable for group of words to be stored in the tree.

3.4 Multi-way Tree

For our application a 56-way tree (especially for Gurmukhi Script) with a route for each letter at every node has been implemented. In multi-way tree, searching is simple, efficient and fast, as is its construction, however the memory overhead is large. So, in order to reduce this memory overhead, we have used linked list instead of static array of 56 pointers. To make searching efficient and fast, we have used bitfields. So, each node of the tree is represented by the following structure:

```
RECORD Node
    BEGIN
        Bitfield    :   BitfieldType
        Splitchar   :   CharacterType
        Flag        :   Boolean
        NextPtr     :   Pointer to LinkedListType
    END

RECORD LinkedListType
    BEGIN
        InfoPtr     :   Pointer to Node
        LinkPtr     :   Pointer to LinkedListType
    END
```

So, instead of having 56 pointers to indicate the next letters in word, this method uses a 56 bitfields, out of which the bits are set (i.e. = 1) if that letter is allowable, and not (i.e. = 0) if that letter is not allowable, and corresponding entry of node, holding that letter, is made in the linked list. The *Splitchar* and Flag fields are similar to as mentioned earlier. The next pointer will be used to store the address of *LinkedListType* structure.

In the structure *LinkedListType*, *InfoPtr* pointer will be used to store the address of node and *LinkPtr* pointer will be used to store the address of next *LinkedListType* structure.

3.5 Reduced Memory Method Tree

The alternative data structure, devised by C. J. Wells *et.al.*[8], which includes the advantages of the 56-way tree method, but reduces memory considerably, was adapted for Punjabi dictionary. Each node in a reduced memory method tree is represented by the following structure

```
RECORD Node
    BEGIN
        Bitfield      :   BitfieldType
        Flag          :   Boolean
        NextArrayPtr  :   Pointer to Array of Node Pointer
    END
```

Here, in this structure, node does not contain the character type field as in case of multi-way tree. Bitfield is of 56 bits similar to the multi-way tree. The purpose of bitfield is also same. Each node also has one *NextArrayPtr* pointer which, if used, points to a variable length array of pointers to nodes (i.e. 56 bits + 1). The field named flag is a boolean field will have value 'TRUE' if the character is the last one in the word.

When searching for a particular string, it is apparent immediately at any node whether the required route from that node exists or not by checking the relevant bit of the 56 flags. If it is set, that route is followed. This means counting how many of 56 bits are set up to and including the required one, to establish which member of the pointer array to the follow to next level in the tree.

4 Results and Conclusion

We have implemented five data structures i.e. binary search tree, trie, ternary search tree, multi-way tree, and reduced memory method tree for a Punjabi dictionary [9] of twenty six thousands words. The results of experiment done on these implementations in context of memory used, time taken for successful and unsuccessful search, and all other factors are shown in Table 1. These results are obtained on a Compaq Deskpro Pentium II with 64 Mb of main memory. For experiment a text of 20,000 words was used, which consisted of 10,000 valid and 10,000 invalid words.

It is observed that Binary Search Tree is the most suitable data structure in terms of memory consumed and time taken for successful as well as unsuccessful search for words. The limitation of Binary Search Tree is that it is very difficult to offer a suggestion list or to look up all words differing by 1 or 2 characters. This limitation can be removed by using Trie structure. In case of trie, too, the trie in which each node contains the information of the Laga and Lagaakhar associated with each consonant is more memory efficient than the trie in which each consonant, Laga and Lagaakhar are separately stored at each node. The processing time for both tries was same for successful as well for unsuccessful searches. The ternary search tree was more memory efficient than the trie in which each node stored one character, but is inferior to that trie in terms of searching speed. The multi-way tree is the bulkiest and slowest data structure while reduced memory tree also does not provide much improvement, the reason being that in case of Punjabi the tree is 56 way as compared to 26 way tree in English ans thus space is need and to store all 56 possible paths and consequently more time is taken to search and traverse the correct path.

Tree Type	Memory used(Mb)	Successful searches on 10,000 words		Unsuccessful searches on 10,000 words	
		No. of nodes searched	Search time (sec)	No. of nodes searched	Search time (sec)
Binary Search Tree	0.4	316,668	7	140,192	8
Trie (storing syllable in node)	0.5	1,940,342	7	1,241,267	7
Trie(storing character in node)	0.8	2,041,754	7	1,436,338	7
Ternary Search Tree	0.6	228,912	9	78,684	9
Reduced Memory Method Tree	0.8	54,989	10	29,222	9
Multi-way Tree	1.3	441,754	10	85,914	10

Table 1 : Memory requirements and processing times for different types of trees

5. References

[1] D. E. Knuth, "Digital searching", The Art of Computer Programming, Addison-Wesley, Reading, Massachusetts, vol. 3, 1973, pp. 481-499.

[2] Karen Kukich, "Techniques for automatically correcting words in text", ACM Computing Surveys, vol. 24, no. 4, 1972, pp. 377-439.

[3] H. Takahashi, N. Itoh, T. Amano, and A. Yamashita, "A spelling correction method and its application to an OCR system", Pattern Recognition, vol. 23,no.3/4, 1990, pp. 363-377.

[4] J. L. Peterson, Computer Programs for Spelling correction, Springer-Verlag, Berlin(1980).

[5] E. Fredkin, "Trie Memory", Commun. of the ACM, vol. 3, no. 9, 1960, pp. 490-500.

[6] Sukhjeet Kaur, Development of spell checker with grammar and electronics dictionary for Punjabi word processor, An M.Tech. thesis submitted to Dept. Of comp. Sc. & Eng., Punjabi University, Patiala, 1997.

[7] Jon Bentley and Bob Sedgewick, "Ternary search trees", Dr. Dobb's Journal, April 1998, pp. 20-25.

[8] C. J. Wells, L. J. Evett, P.E. Whitby and R. J. Whitrow, "Fast dictionary look-up for contextual word recognition", Pattern Recognition, vol. 23, no. 5, 1990, pp. 501-508.

[9] Punjabi-English Dictionary, Publication Bureau, Punjabi University, Patiala, 1994.