LZ compression of static linked list tries

Strahil Ristov[1], Damir Boras[2] and Tomislava Lauc[2]

[1] Ruđer Bošković Institute, Zagreb, Croatia [2] Faculty of Philosophy, University of Zagreb, Croatia

A variant of LZ (Lempel–Ziv) procedure for compressing static trie structures is investigated for different types of natural language data. A compressed trie is produced from the set of 40,000 bilingual dictionary entries and the results are compared to previous results for a set of 750,000 different word forms. A possible use of a sliding window in LZ procedure for linearization of algorithm execution time is investigated. The results show considerable application potential.

Keywords: natural language processing, dictionary, LZ data compression, trie data structure.

1. Introduction

Compressed trie structures are often used for storing huge amounts of static natural language data, such as dictionaries, thesauri or simple lists of words or word phrases (Purdin, 1990), (Morimoto et al., 1994). For an alphabet of M letters, a trie is a M-way tree in which each node represents a string prefix that is obtained by following the unique path from root node (Fredkin, 1960), (Knuth, 1973). Identical prefixes of different strings are represented with only one node. Different strings are obtained by following different links from the prefix node. With the search time proportional to the length of a word, a trie structure is very fast to search but it can be huge. When storing words of a natural language, and especially an inflected one as Croatian, it is convenient that a node comprises only one letter. A full M-way tree would then have L^M nodes, where L is the length of the longest entry. An example of such a sparse tree for a three letter alphabet and $L = 3$ is presented in Figure 1. In such a tree most of the nodes would be empty and they can be excluded. The simplest way is to implement the trie using linked list instead of M-dimensional nodes. Alternatively, or additionally, various trie compression procedures can reduce the size achieving considerable savings in space. The typical trie compression method is a transformation into a minimal finite automaton for a given finite language (finite set of words) or a directed acyclic word graph (DAWG). For a set of natural language words a DAWG is a very practical representation, compact and fast (Appel and Jacobson, 1988) (Lucchesi and Kowaltowski, 1993). It has an important advantage in that there exists a linear algorithm for automata minimisation (Revuz, 1992), but it has a drawback in that it is mostly appropriate for storing simple word lists and is therefore restricted mostly to the spelling correction applications. DAWG cannot be used efficiently for storing more complex entries like dictionary entries or word phrases that have greater inner diversity.

Croatian is a highly inflected language and each word lemma (i.e. a noun in nominative singular, a verb in infinitive) has on the average 10 distinct word forms for different tenses, cases or gender. The grammatical rules for word form production are too complex and there are too many exemptions for implicit knowledge based storage of a set of valid Croatian words. An explicit list is therefore better suited for computer applications. In (Ristov, 95) it was demonstrated how a compressed trie can be successfu lly used for storing a huge list of Croatian word forms. Some 750,000 different words (word forms) were stored in a structure less than 300 KB in size, and the access speed is measured in thousands of words per second on an average PC. The algorithm used was more elaborate than automata minimisation; the method of compression can be described as a variant of LZ compression applied to a linked list implementation of a trie (Ristov, 1997) and is brie fly

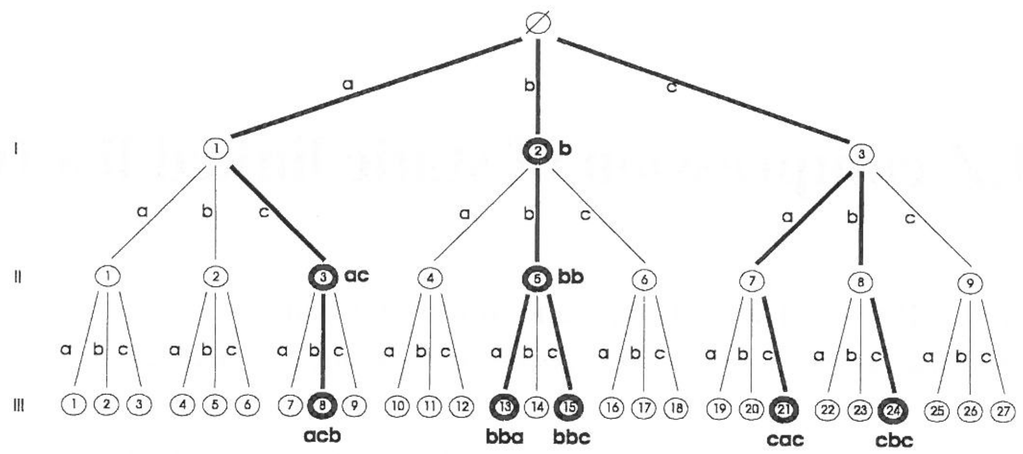

Fig. 1. Four levels (including the zero level root node) of ternary trie for an alphabet (a, b, c). Trie comprises words: ac, acb, b, bb, bba, bbc, cac, cbc. Thick lines indicate existing transitions and final states. Of 27 nodes only eight are used.

outlined in Chapter 2 of the paper. In Chapter 3 we show how this trie compression method can be applied to a set of a more complex entries, namely a Croatian-English dictionary of economics terms and phrases. By comparing results with those for the set of word forms we extrapolate for bigger complex data sets. In Chapter 4 we present some execution time considerations. Since, unlike DAWG construction, LZ compression has inherently quadratic time complexity, we investigated possibilities of a speedup along standard LZ window speedup schemes. We conclu de in Chapt er 5 that presented algorithm and data structure show a considerable application potential.

2. Linked list trie compression

The principle of LZ linked list trie compression is demonstrated in Figure 2 for the set of four words. Figure 2a) presents the construction of a corresponding trie with one character per transition. Empty nodes and nonexistent transitions are omitted. Thicker circles represent terminal nodes. Actual linked list implementation is shown in Fig. 2.b). Each element of the list consists of four parts:

1) a character;

2) a marker for the end of a word, indicated by the thick circles;

3) a pointer to the next character in the word (null if there is no continuation of the word), represented with horizontal arrows in Fig. 2b);

4) a pointer to the next element on the same level in the node, represented with the inflected pointing arrows on Fig. 2b).

The linked list itself is a way to exclude nonexistent nodes, and the actual implementation of Fig. 2b) gives rise to a possibility for a further reduction in the number of elements. With the trie structure presented in this way it is apparent that identical subsequences of elements can be replaced with pointers in a LZ manner. LZ is a generic name for data compression methods where, in a stream of data, repeated sequences are replaced with pointers to their previous occurrences. The name originates from the se minal work (Ziv and Lempel, 1977). There exist a variety of different implementations with the only fixed common point being that pointers occupy less space than original data. An exhaustive overview of LZ methods can be found in (Bell et al, 1990).

In Fig. 2c) repeated sequences of elements in structure of Fig. 2b) are replaced with two different types of pointers; we shall call them one-way and two-way pointers. The one-way pointer, represented with ⧅ sign, indicates that the reading of the structure should continue from the element at which the pointer points. The sequences of elements that are replaced with one-way pointers are identical subtrees and, in automata terminology, the effect of one-way pointers is the merging of equivalent states. Two-way pointers, represented with ⊠ sign, replace identical subsequences of elements that do not belong to identical subtrees. This means

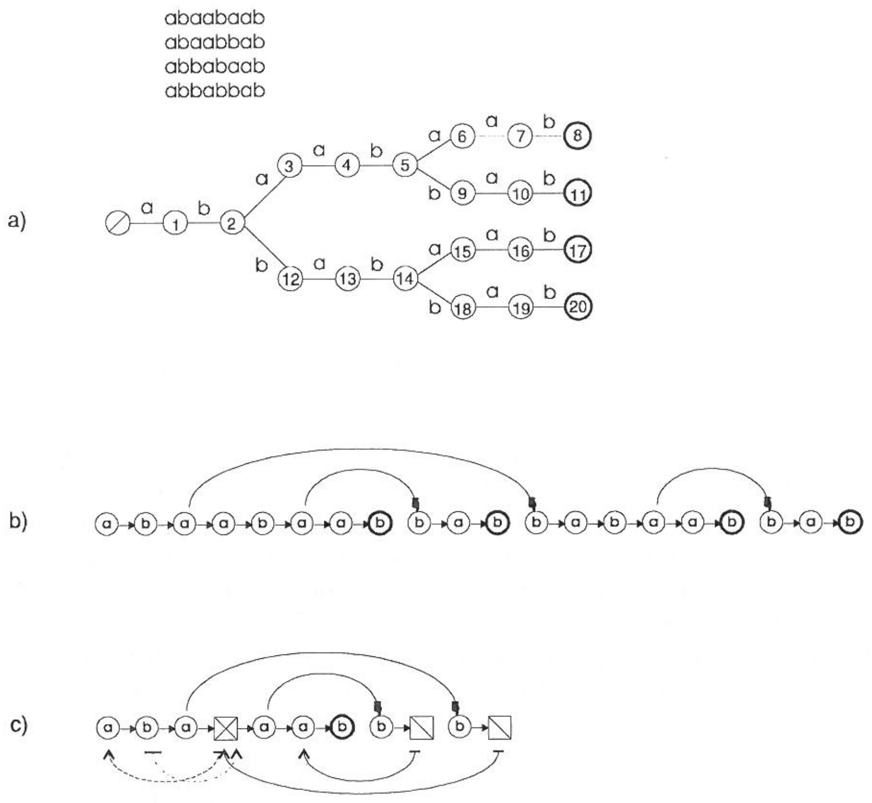

Fig. 2. The construction of the compressed linked list trie for a set of 4 words with binary alphabet (a, b).

that only a certain number of elements should be followed starting from the element at which the two-way pointer points, and that after that number the reading should be continued back from the position of the pointer. An additional information — the number of elements replaced, is associated with each two-way pointer. In Fig. 2c) the arrows below the elements, originating at pointers, indicate to which elements the pointers point. Full lines are for the one-way pointers. The dashed line points to the beginning of the subsequence replaced by the two-way pointer and the dotted line indicates that after two elements the reading should continue from the pointer position. In order to replace every (existing) repetition with a pointer, a double loop is used to check each subsequence for equivalence against every other. The resulting structure can have multilevel nested pointer paths, i.e. a pointer can point at the subsequence containing another pointer. The LZ structure compression obtained in this way is very efficient. However, the quadratic time complexity of a double loop becomes restrictive for larger data sets.

3. Data sets: description and the compression results

Two different data sets were used to test the performance of the described compression method. One was the set of some 750,000 different Croatian word forms, and the second a set of 40,000 Croatian-English economics dictionary phrases. A detailed description and compression results for the word forms set are presented in (Ristov, 1995 and 1997). The average word length is approximately 10 characters and the set consists of some 68,000 different word lemmas with an average number of about 11 word forms per lemma. As an example, the 7 forms of one lemma are presented in Fig. 3a). In ASCII format the set size is over 8Mb. When compressed, and with optimised element size, it is under 300Kb. For this set the size of LZ compressed trie is around 30% less than that of

pismohrana
pismohranama
pismohrane
pismohrani
pismohrano
pismohranom
pismohranu

a)

Dow Jonesov indeks — Dow Jones indeks
Dow–Jonesov industrijski prosjek — Dow Jones industrial average
Europska ekonomska komisija (ECE) — Economic Comission for Europe
Europska ekonomska zajednica (EEZ) — European Economic Community (EEC)

b)

Fig. 3. The typical examples of entries in two data sets.

DAWG. For some cases it is arguable whether this saving in space justifies longer compression time. The time sacrifice is justified in applications where the compressed data structure is produced only once and is used for a long time or is widely distributed. On the other hand, for applications where data structures are produced at the run time or changed often, a faster, although less efficient procedure should be preferred.

A typical example of few entries in the second set is shown in Fig. 3b). The entries are longer — around 40 characters per entry on average, and the ASCII size is 1,6 Mb. It is obvious that there are much less substring repetitions than in the first set. Also, there are almost no entries with longer identical endings, so a DAWG would be of approximately the same size as a trie. However, since there are enough repetitions inside the strings, the LZ trie compression was attempted. Results are presented in Table 1 for subsets of original sets with ascending number of entries. For the word forms, subsets were obtained based upon a random choice of lemmas. At the beginning we randomly selected 10,000 lemmas from the original 68,000. From those we again randomly chose subsets of 1000, 100, and 10 lemmas such that smaller sets were subsets of larger ones. Finally, experimental sets were produced by expanding lemmas into all of their word forms, hence an odd numbers of entries. The dictionary entry su bsets were produced straightforwardly by random selection, again with smaller sets being parts of larger ones. It is obvious that the tries constructed from higher number of entries compress better. This was expected since the variety of substrings in natural languages is limited and, as the number of entries grows, more and more substrings of new entries are found in previous strings. When fitting these few values of number of entries against the size of compressed structure, we got normalized logarithmic functions of

$(-6,6+\ln x)$ for word forms, and

$(-5,8+\ln x)$ for dictionary entries.

The function curve slopes are quite similar, so a valid projection for larger sets of complex phrases can be deducted from the results for word forms. The actual values of dictionary compression results prove that LZ compressed

	word forms					dictionary entries			
no. of entries	120	1,235	11,672	113,736	751,519	100	1,000	5,000	40,000
ASCII size (byte)	1,452	14,711	136,656	1,328,776	8,276,599	5,582	43,453	182,259	1,646,349
trie size (elements)	210	2,185	19,634	176,845	1,029,669	4,847	31,967	118,512	1,054,057
compressed trie size (elements)	143	750	4,134	25,696	97,001	2,089	10,657	35,138	239,4586

Table 1. The trie compression results for sets of different sizes for two data types.

	word forms		dictionary entries	
	no. of elements in trie = 1,029,669		no. of elements in trie = 1,054,057	
W	N_c	T	N_c	T
1000	355,176	5'	576,587	5'
2000	295,120	9'	534,860	18'
5000	239,505	17'	486,877	37'
10,000	206,766	25'	454,638	55'
20,000	181,146	44'	423,009	1h 40'
50,000	154,854	1h 28'	382,155	3h 58'
100,000	138,513	2h 30'	348,223	6h 40'
unbounded	97,003	7h 40'	239,458	21h 30'

Table 2. Tradeoffs between the size (in number of elements) N_c of compressed structure and the compression time T, for various window sizes W.

trie is a plausible structure for storing and accessing such type of data. After the compression the size of the elements can be optimised to 3 to 4 bytes, so all 40,000 dictionary entries can be stored in about 800Kb. The access speed is high as with any LZ compressing scheme; a typical access speed is over 1000 words per second on a 100MHz PC.

4. Window speedup

It has been proved that full LZ compression asymptotically achieves maximum entropy of compressed data (Ziv and Lempel, 1977). In describing the procedure of LZ compression a notion of a sliding comparison window is used. In a stream of symbols each symbol sequence is compared to the previous symbols inside the given interval called window. When the size of the window is unbounded, asymptotically optimal compression is achieved, but the time complexity of the procedure is $O(N^2)$, where N is the number of input symbols. If the size of the window is limited to a constant W, then the symbol at the current position in a stream of data is compared only to the W preceding symbols. In this way, a compression algorithm has linear time complexity $O(N * W)$, but the compression is less efficient and depends on the size of the window and on the type of data. The dependencies of compression efficiency on the size of the window have been thoroughly investigated for various file compression algorithms (Bell et al., 1990), but, to our know ledge, so far no one has investigated this tradeoffs in compressing searchable tree structures. We have conducted an experiment with the described data sets and different window sizes. The results are presented in Table 2. Time data would depend on the processor used and should be regarded as a relative information; these values were obtained on a PC486/100MHz. Compression was performed with various window sizes, from $W = 1,000$ to $W = 100,000$. For both data sets even a very big W of about one tenth of initial input trie causes over 40% increase in the size of the output compressed structure. On the other hand, modest size windows produce structures not more than twice the size of that with an unbounded window. A good compromise values would be $W = 10,000-20,000$ for the word forms set, and $W = 5,000 - 10,000$ for the dictionary set. For these values of W the compressed structure is not bigger than twice the size of the optimally compressed one, but the compression time is greatly reduced and, more important, linearized. Overall longer times for dictionary entries are due to the shorter repeated element sequences in trie for that set.

5. Conclusion

By employing an original LZ compression method on static tries, we stored 40,000 complex dictionary entries in a structure that has a very fast access time, and has the size of about half of the full ASCII size. By investigating the method performance on different data types we infer that larger dictionaries would compress even better. For really large data sets the restrictive factor is the quadratic execution time of the compression procedure. This can be overcome by limiting the LZ window to a constant value. In this way the execution time becomes linear, but the compression performance gets worse. The reasonable compromise seem to be the window sizes, where the final structure is not more than twice the size of the optimally compressed structure. For very large data sets we expect very efficient compression, so that this increase in size should still be acceptable. We conclude that this compression method for searchable data can be efficiently utilised for any huge sets of static natural language data like encyc lopaedias, thesauri or dictionaries of any sort.

References

[1] A. W. APPEL, G. J. JACOBSON, The world's fastest scrabble program, *Communications of the ACM*, Vol. 31, No. 5. (1988)

[2] T. BELL, J. G. CLEARY, WITTEN, I. H., *Text Compression*, Prentice–Hall, Englewood Cliffs, 1990.

[3] E. FREDKIN, Trie memory, *Communications of the ACM*, Vol. 3, No. 9, (1960), 490–499.

[4] D. E. KNUTH, *The Art of Computer Programing*, Vol. 3: Sorting and Searching, Addison–Wesley, 1973.

[5] C. L. LUCCHESI, T. KOWALTOWSKI, Applications of finite automata representing large vocabularies, *Software-Practice and Experience*, Vol. 23, No. 1, (1993), 15–30.

[6] K MORIMOTO, H. IRIGUCHI, J. AOE, A method of compressing trie structures, *Communications of the ACM*, Vol. 24, No. 3, (1994), 265–288.

[7] T. D. M PURDIN, Compressing tries for storing dictionaries, *Proceedings of the 1990 Symposium on Applied Computing*, (1990) Fayetteville, AR.

[8] D. REVUZ, Minimisation of acyclic deterministic automata in linear time, *Theoretical Computer Science*, Vol. 92, No. 1, (1992), 181–189.

[9] S. RISTOV, Space saving with compressed trie format, *Proceedings of the 17th International Conference on Information Technology Interfaces*, (1995) Pula.

[10] S. RISTOV, Metoda analize i iskorištenja zalihosti u konačnom skupu znakovnih nizova s primjenom na hrvatski obličnik, PhD Thesis in Croatian, University of Zagreb, 1997.

[11] J. ZIV, A. LEMPEL, A universal algorithm for sequential data compression, *IEEE Transactions on Information Theory*, Vol. IT–23, No. 3, (1977), 337–343.

Received: October, 1997
Accepted: December, 1997

Contact address:
Strahil Ristov
Ruđer Bošković Institute
Zagreb
Croatia
e-mail: ristov@olimp.irb.hr

Damir Boras and Tomislava Lauc
Faculty of Philosophy
University of Zagreb, Croatia
e-mail: dboras@ffzg.hr
e-mail: tzubrini@ffzg.hr

STRAHIL RISTOV received his B.S. degree in engineering and M.S. and Ph.D. degrees in computer science from Faculty of Electrical Engineering and Computing, University of Zagreb. He has been with the Laboratory for Stochastic Processes at Ruđer Bošković Institute since 1991. His research interests include image processing and pattern recognition, natural language processing, data structures and compression. He is author and coauthor of seven papers.

DAMIR BORAS received his B.S. degree in electrical engineering from the Faculty of Electrical Engineering and Computing, University of Zagreb. Currently he is working on Ph.D. thesis in the field of text segmentation in Croatian language at the Department of Information Science, Faculty of Philosophy, University of Zagreb, where he teaches text processing, lexical data bases and computer networks. His interests include lexical data bases, corpuses and text processing in Croatian. He has published over 30 papers in the field of information science and is coauthor of two textbooks for information and computer science.

TOMISLAVA LAUC received her B.S. and M.S. degrees from Faculty of Philosophy, University of Zagreb. She is a research assistant at the Department of Information Science at that faculty and also a member of the research team on the project "Models of Knowledge and Natural Language Processing". She graduated with the master thesis entitled "Possibilities of Tagging of a Croatian Text Corpus" and has published several papers concerning NLP.