

Reducing cache misses in n -gram methods using de Bruijn sequences

George Coulouris

25 November 2008

Abstract

A de Bruijn sequence contains all possible n -grams as unique overlapping substrings. Such sequences have a wide variety of applications in areas including cryptography, image processing, and robotics. Efficient algorithms exist to construct such sequences. Given an efficient algorithm for locating an n -gram within a de Bruijn sequence, reordering array indices can improve the cache properties of n -gram methods.

Introduction

A de Bruijn sequence contains all possible n -grams as unique overlapping substrings. For example, the string “aacagatccgctggtta” contains all possible 2-grams over the alphabet “acgt”. The existence and properties of such sequences over general alphabets were first proved by de Bruijn in 1946, although work on binary alphabets dates to 1894 [1]. It was later shown that, for an alphabet of size k , there are $\frac{(k!)^{k^{n-1}}}{k^n}$ de Bruijn sequences. A linear feedback shift register implementation for constructing de Bruijn sequences was first proposed by Fredericksen [2] in 1968, based on the work of Golomb in 1967. Lempel provided a recursive construction in 1970 [3]. de Bruijn sequences can be used to generate pseudorandom numbers for the keying of stream ciphers [4].

The decoding problem

The problem of locating a particular n -gram within a de Bruijn sequence is known as the de Bruijn decoding problem. In general, this problem is computationally difficult. An obvious solution is to create a mapping from lexicographic ordering to de Bruijn ordering, but this is not practical for large values of n . One can trade space for time through the use of algorithms similar to Shanks’ algorithm for discrete logarithms [5], but this does not provide an asymptotic improvement. Efficient decoding algorithms do exist, however, for special cases of recursively-constructed sequences, requiring only $O(n \log n)$ time [6]. The decoding property can be used to effect one dimensional position sensing in

robotics applications, and can be extended to two dimensions to yield a de Bruijn torus [7].

***n*-gram methods**

n-gram methods are widely used in many areas, including natural language processing, bioinformatics, and cryptanalysis. A well-known example in cryptanalysis is the Kasiski test [8], used to infer the key length in Vigenère ciphers. In this context, we wish to construct a mapping from all possible *n*-grams to the locations where they occur in a given piece of ciphertext. A common approach to implementing this mapping is through the use of a lookup table. When dealing with *n*-grams over an alphabet of size *k*, we construct a table of k^n cells corresponding to each of the *n*-grams in lexicographic order. Each cell contains a pointer to a list of offsets where that *n*-gram occurred.

After generating such a structure, it is often desirable to perform lookups against it to search for occurrences of substrings. For example, one might first build a lookup table based on a set of keywords of interest and then use it to scan an incoming stream of text for instances of those keywords. This is implemented by constructing successive *n*-grams out of the input stream, going to the corresponding cell in the lookup table, and checking to see if it occurred in the original set. If so, the location is flagged as a possible match and the scanning proceeds.

If we view such an input stream as a shift register of width *n* and assuming that scanning proceeds one character at a time, we immediately notice that successive instances of the shift register are highly correlated, as they share $n - 1$ of the same characters in the same order. If these successive instances are viewed numerically, however, their numerical value varies widely, since the most significant digit is likely to change. Since these numerical values are used as indices to the lookup table, successive accesses are “far apart” in computer memory, leading to poor performance in cache-based computer architectures. Indeed, in biological applications [9], it is common to use tables containing 4^{12} elements. Such a table occupies 64 megabytes of memory, greatly exceeding the size of contemporary CPU caches. We are thus faced with the problem of accessing a larger-than-cache array in a manner that is effectively random. In order to improve performance, we must either compress the array, or make the access patterns less random.

Visual inspection of de Bruijn sequences generated by Fredricksen’s algorithm shows that substrings of length $n - 1$ occur “close” to one another within the sequence. Similar to how other data structures have been reordered for better cache properties [10], I hypothesize that de Bruijn sequences can be used to reorder *n*-gram array lookups for better cache performance.

In order to implement such a reordering, rather than using the raw *n*-gram from the input stream as an index to the lookup table, we must instead find where that *n*-gram would occur in a de Bruijn sequence. Assuming a black box that solves the de Bruijn decoding problem described above, the hope is that

successive overlapping n -grams in the input will map to nearby elements after de Bruijn decoding, where they were far apart in the ordinary lexicographic ordering.

Methods

To test this hypothesis, I created one lookup table using the traditional lexicographic ordering and another lookup table using a de Bruijn ordering created with Fredricksen's algorithm. I then generated a set of 65537 random 8-grams over an alphabet of size 4. I resolved this set against each of the lookup tables and recorded the string of addresses generated. For each lookup table, I then examined the the absolute value of the differences of successive addresses. I used the mean absolute difference as a metric of cache friendliness, with lower numbers being better.

Results

This experiment yielded $n = 65536$ address deltas for each lookup table. The lexicographic ordering yielded a mean absolute difference $\mu = 19058$ with $\sigma = 13044$ and the de Bruijn ordering yielded a mean absolute difference of $\mu = 5703$ with $\sigma = 9490$.

Conclusion

Using a de Bruijn sequence to reorder array indices significantly reduces the mean absolute difference in the memory addresses generated by n -gram lookup techniques. As of this writing, this result requires one de Bruijn decoding operation per lookup. While efficient de Bruijn decoding algorithms exist, it is likely that a hardware implementation would be required in order to realize practical performance gains in in-core applications; a software implementation may suffice for out-of-core applications.

Future work

It would be instructive to test different values of n and k to determine if there is an empirical bound on the improvement in mean absolute address difference. It is also not known if different de Bruijn sequence construction techniques will significantly alter the locality properties of the sequence and hence the efficacy of this technique.

References

- [1] N.G. de Bruijn. Acknowledgement of priority to c. flye sainte-marie on the counting of circular arrangements of $2n$ zeroes and ones that show each n -letter word exactly once. Technical report, Technological University Eindhoven, 1975.
- [2] H Fredricksen. The lexicographically least de bruijn cycle. *J. Combin. Theory*, (9):1–5, 1970.
- [3] A. Lempel. On a homomorphism of the de bruijn graph and its applications to the design of feedback shift registers. *IEEE Trans. Comput.*, 19(12):1204–1209, 1970.
- [4] C G Gunther. Alternating step generators controlled by de bruijn sequences. In *Advances in Cryptology–EUROCRYPT ’87 (LNCS 304)*, pages 5–14, 1988.
- [5] Chris J. Mitchell, Tuvit Etzion, Member Ieee, Member Ieee, and Kenneth G. Paterson. A method for constructing decodable de bruijn sequences. *IEEE Transactions on Information Theory*, 42:1472–1478, 1996.
- [6] Jonathan Tuliani. De bruijn sequences with efficient decoding algorithms. *Discrete Math.*, 226(1-3):313–336, 2001.
- [7] Glenn Hurlbert and Garth Isaak. On the de bruijn torus problem. *J. Comb. Theory Ser. A*, 64(1):50–62, 1993.
- [8] Douglas R. Stinson. *Cryptography: Theory and Practice*.
- [9] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, October 1990.
- [10] H. Prokop. Cache-oblivious algorithms, 1999.