Improved Bounds for Dictionary Look-up with One Error

Gerth Stølting Brodal

BRICS, Department of Computer Science, University of Aarhus, DK-8000 Århus C, Denmark. gerth@brics.dk. [1]

S. Venkatesh [2]

School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai 400005, India. venkat@tcs.tifr.res.in.

Abstract

Given a dictionary S of n binary strings each of length m, we consider the problem of designing a data structure for S that supports d-*queries*; given a binary query string q of length m, a d-query reports if there exists a string in S within Hamming distance d of q. We construct a data structure for the case $d = 1$, that requires space $O(n \log m)$ and has query time $O(1)$ in a cell probe model with word size m. This generalizes and improves the previous bounds of Yao and Yao for the problem in the bit probe model. The data structure can be constructed in randomized expected time $O(nm)$.

Key words: Data Structures, Dictionaries, Hashing, Hamming Distance

1 Introduction

Minsky and Papert in 1969 posed the following problem, that has remained a challenge in data structure design [9].

> Let S be a set of n binary strings of length m each. We want to construct a data structure for S that supports fast d-queries; that is, given a binary

[1] BRICS (Basic Research in Computer Science), a Center of the Danish National Research Foundation

[2] This work was done while visiting BRICS at the University of Aarhus.

query string q, determine if there is a string in S within Hamming distance d of q.

To date, no efficient solutions are known for this problem for arbitrary n, m and d. Dolev *et al.* [3,4] and Greene, Parnas and Yao [6] made some progress on the problem for the case when d is large. Manber and Wu [8] considered applications to password security and spellchecking of large files.

The theoretical study of the problem for small d was started by Yao and Yao [12], who considered the case $d = 1$ and presented a scheme in the bit probe model that uses space $O(mn \log m)$ bits and queries requiring $O(m \log \log n)$ bit probes. Brodal and Gąsieniec [1] considered the problem for the case $d = 1$ in the standard unit-cost RAM model with logarithmic word size, and presented a data structure of size $O(mn)$ words supporting 1-queries in $O(m)$ memory accesses.

In this paper, we also consider the case $d = 1$. We give a scheme proving the following result.

Theorem 1 *In a cell probe model with word size m, a data structure exists that achieves query time $O(1)$ using space $O(n \log m)$. It can be constructed in randomized expected time $O(nm)$.*

Translated to the bit probe model, this yields a query time $O(m)$ and space $O(mn \log m)$ scheme. Our scheme improves the query time of Yao and Yao by a factor $O(\log \log n)$. Our scheme can be implemented on a standard unit-cost RAM which is not true for the data structure of Yao and Yao since it is proved in the bit probe model. It is unknown if the data structure of Yao and Yao can be implemented on a standard unit-cost RAM or in the cell probe model with word size m such that queries are supported in time $O(\log \log n)$ [12].

The model of computation we use is the *cell probe* model with word size m defined in [11]. In this model, each word is assumed to be m bits long. The *space* used by a scheme is measured as the number of words used by the storing scheme. The query algorithm answers queries by making adaptive cell probes and *time* is counted as the number of cell probes made by the query algorithm. The bit probe model is a cell probe model with word size 1.

2 The scheme

To describe our scheme, we will make use of the following results by Dietzfelbinger [2] and Jacobs and van Emde Boas [7].

Theorem 2 (Dietzfelbinger) *Given a set S of n binary strings each of m*

bits in a cell probe model with word size m, there exists a data structure using $O(n)$ cells that supports membership queries of S in time $O(1)$. The construction time of the data structure is randomized expected $O(n)$.

The result of Dietzfelbinger is a simplification of the data structure of Fredman, Komlós and Szemeredi [5]. It eliminates the requirement for knowing a prime in the range $[2^m \mathinner{..} 2^{m+1}]$. In the following we refer to the data structure of Theorem 2 as a FKS data structure.

Definition 3 *A function $h : \{0, 1, \ldots, \ell - 1\} \to \{0, 1, \ldots, k - 1\}$ is* perfect *for $S \subseteq \{0, 1, \ldots, \ell - 1\}$ if h is 1-1 on S. A family H is an (ℓ, n, k)-family of perfect hash functions if for all $S \subseteq \{0, 1, \ldots, \ell - 1\}$ of size n, there is an $h \in H$ that is perfect for S.*

The question of representing efficiently families of perfect hash functions is well studied. Schmidt and Siegel [10] described a $(2^m, n, O(n))$-family of perfect hash functions where each hash function can be represented by $\Theta(n + \log m)$ bits. Jacobs and van Emde Boas [7] gave a simpler solution requiring $O(n \log\log n + \log m)$ bits in the standard unit-cost RAM model augmented with multiplicative arithmetic, that suffices for our purposes. The construction in [7] makes repeated use of the data structure in [5] where some primes are assumed to be known. By replacing the applications of [5] by applications of [2], the randomized construction time in Theorem 4 follows immediately.

Theorem 4 (Jacobs and van Emde Boas) *There is a $(2^m, n, O(n))$-family of perfect hash functions H such that any hash function $h \in H$ can be represented in $\Theta(n \log\log n + \log m)$ bits and evaluated in $O(1)$ time. The perfect hash function can be constructed in randomized expected $O(n)$ time.*

Our scheme is based on the following simple idea: If the query string exactly matches with one of the strings in S, then a FKS data structure can be used to check this efficiently. If the query string differs from some string in S by a single bit, then it is sufficient to find such a bit position. We use the Jacobs-van Emde Boas scheme to do this. Given the bit position, we can then flip this bit in the query string and use the FKS data structure for an exact search to verify that the query is within Hamming distance 1 of a string in S.

Let $N(S)$ denote the set of all strings at Hamming distance one of a string in S. Note that $|N(S)| \leq mn$. For any string $x \in N(S)$, we denote by *index*(x), the position of the bit that was flipped in a string from S to obtain x. If there is more than one choice for *index*(x), we choose one arbitrarily.

Storing scheme for a set S.

(1) Construct a FKS data structure for S.

(2) Using the Jacobs-van Emde Boas scheme (Theorem 4), construct a perfect hash function h for $N(S)$ and store h.
(3) Construct a look-up table T of size $O(|N(S)|)$ where each entry is $\lceil \log m \rceil$ bits. For $x \in N(S)$, let $T[h[x]] = index(x)$.

Query scheme for a query string q.

(1) Check if $q \in S$ using the FKS data structure for S. If $q \in S$, report *yes*.
(2) Otherwise, evaluate $h(q)$ and let $i = T[h(q)]$.
(3) Flip the ith bit of q and report *yes* if the new string is in S using the FKS structure for S. Otherwise report *no*.

That the scheme works correctly follows from the discussion above. From Theorem 2 and Theorem 4 it follows that in a cell probe model with word size m, the scheme described above uses $O(n \log m)$ cells, and answers queries in $O(1)$ cell probes, and Theorem 1 follows.

Acknowledgment

The authors wish to thank Peter Bro Miltersen and Rasmus Pagh for helpful discussions.

References

[1] G. S. Brodal and L. Gąsieniec. Approximate dictionary queries. In *Proc. 7th Combinatorial Pattern Matching*, volume 1075 of *Lecture Notes in Computer Science*, pages 65–74. Springer Verlag, Berlin, 1996.

[2] Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *13th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1046 of *Lecture Notes in Computer Science*, pages 569–580. Springer Verlag, Berlin, 1996.

[3] D. Dolev, Y. Harari, N. Linial, N. Nisan, and M. Parnas. Neighborhood preserving hashing and approximate queries. In *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 251–259, 1994.

[4] D. Dolev, Y. Harari, and M. Parnas. Finding the neighborhood of a query in a dictionary. In *Proc. 2nd Israel Symposium on Theory of Computing and Systems*, pages 33–42, 1993.

[5] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the Association for Computing Machinery*, 31(3):538–544, 1984.

[6] D. Greene, M. Parnas, and F. Yao. Multi-index hashing for information retrieval. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 722–731. IEEE Computer Society Press, 1994.

[7] C. T. M. Jacobs and P. van Emde Boas. Two results on Tables. *Information Processing Letters*, 22(1):43–48, 1986.

[8] U. Manber and S. Wu. An algorithm for approximate membership checking with application to password security. *Information Processing Letters*, 50(4):191–197, 1994.

[9] M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, Mass., 1969.

[10] J. P. Schmidt and A. Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM Journal on Computing*, 19(5):775–786, 1990.

[11] A. C. C. Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.

[12] A. C. Yao and F. F. Yao. Dictionary look-up with one error. *Journal of Algorithms*, 25(1):194–202, 1997.