The spatial complexity of oblivious k-probe hash functions †

Jeanette P. Schmidt *Alan Siegel*
Ultracomputer Research Laboratory
Courant Institute or Mathematical Sciences
715 Broadway, 10th Floor
New York, NY 10003
Ultracomputer Note #142
June 1988

Abstract

We consider the problem of constructing a dense static hash-based lookup table T for a set S of n elements belonging to a universe $U = \{0, 1, 2, \ldots, m-1\}$. We provide nearly tight bounds on the spatial complexity of oblivious $O(1)$-probe hash functions, which are defined to depend solely on their search key argument. This establishes a significant gap between oblivious and non-oblivious search. In particular, our results include the following:

- Oblivious k-probe hash functions require $\Omega(\frac{n}{k^2}e^{-k} + \log\log m)$ bits.
- A probabilistic construction of a family of oblivious k-probe hash functions that can be specified in $O(ne^{-k} + \log\log m)$ bits, which nearly matches the above lower bound.
- A variation of an $O(1)$ time 1-probe (perfect) hash function that can be specified in $O(n + \log\log m)$ bits, which is tight to within a constant factor of the lower bound.
- Upper and lower bounds for related families of hash functions.

† The work of the first author was supported by DARPA grant F49620-87-C-0065. The work of the second author was supported in part by ONR grants N00014-84-K-0444 and N00014-85-K-0046.

1. Introduction

Hashing is one of the most important and commonly used methods to organize simple collections of information. The applications are extensive, and the subject has a correspondingly rich theoretical literature. In this paper, we will be restricting our attention principally to the dictionary problem, which concerns how to organize a set S of distinct keys within a table T so that the elements can be retrieved quickly. We shall take the data set to be static, so that the hash table need not support insertion or deletion, and consider only open addressing, so that pointers will not be used. Most of our results will focus on 100% utilized tables.

The history of this problem, which we detail in the next subsection, would seem to suggest that virtually all questions have been answered for this specific problem; a variety of lower bounds have been established [Yao81] and [Me84], and elegant constructions have been discovered, which nearly meet the lower bounds [FKS84].

Recently, however, new techniques for organizing data have been devised [FMNSSS88], [FNSS88], which show that an enormous amount of information can be encoded within a search table. The thrust of these results is to show how to exploit non-oblivious search, which can use an adaptive probe strategy based upon information gleaned from unsuccessful probes. The consequences are performance bounds and extensions for the dictionary problem that had been believed to be impossible, for $O(1)$-probe hashing schemes [FNSS88].

The unexpected opportunities demonstrated by these results have lead us to examine afresh the computational models and assumptions underlying the lower bound of [Me84] and the construction of [FKS84]. These results are for 1-probe schemes, which by definition cannot be adaptive. The natural question to ask is whether the opportunity to use the information encoded within the constant number of probed locations is genuinely significant, or if the power of the [FNSS88] scheme is actually due to the ability to query several locations for the search key.

A few preliminary definitions help formalize the problem. We let the data set S comprise n elements belonging to the universe $U = \{0, 1, 2, \ldots, m-1\}$. Our hash-based lookup table T is also of size n. A sequence $H = (h_1, h_2, \ldots, h_k)$ of functions is a **k-probe hash function** for S, if $H : U \mapsto [1, n]^k$, and $T[1..n]$ can be organized so that each item $s \in S$ is located in one of the k probe positions defined by applying the k probe functions to s.

The method of probing must be defined quite precisely. A hashing strategy is **oblivious** if the search locations computed by H, are based solely on the key s, and not on other keys stored in T. In this case, the search strategy is

for $i \leftarrow 1, 2, \ldots, k$ do
    if $T[h_i(s)] = s$ then return$(h_i(s))$
endfor;
if s has not been found then return(FAIL).

In contrast, a **non-oblivious** hashing scheme can make computational use of the keys encountered during unsuccessful search, which offers a wider and conceivably more powerful range of search strategies. Formally, such a hashing scheme differs in that h_i is an i-ary function mapping U^i into $[1, n]$. The non-oblivious search strategy is

for $i \leftarrow 1, 2, \ldots, k$ do
    $s_i \leftarrow T[h_i(s, s_1, s_2, \ldots, s_{i-1})]$;

if $s_i = s$ then return$(h_i(s, s_1, s_2, \ldots, s_{i-1}))$

endfor;

if s has not been found then return(FAIL).

A family $\bar{H}$ is a k-**probe** family for U if every n-element subset $S \subset U$ has some k-probe hash function in $\bar{H}$. A 1-probe hash function for S, is called a **perfect** hash function, and a perfect family for U denotes a corresponding 1-probe family. (In the exposition that follows, we shall frequently take the liberty of suppressing the implicitly understood parameters n, m and even k, for notational convenience.) The principal problem we analyze is how many hash functions are needed to define a k-**probe** family $\bar{H}$, in the oblivious hashing model. In particular, we attain estimates for $\log(\bar{H})$, which is the number of bits needed to specify a particular hash function $H \in \bar{H}$.

1.1 Background

Mehlhorn showed that the bit length of a perfect hash function from S into a table T of size n_1, where $n \leq n_1 < (1-\epsilon)m$, is lower bounded by $\Omega(\, n^2/n_1 + \log\log m)$ [Me82], [Me84]. The explicit construction of efficient perfect hash functions has been explored, among others, by [Me84], [Ma83], [FKS84], [JvE86] and [SvE84]. In [FKS84], an $O(1)$-time computable perfect hashing scheme for full tables (i.e. $n_1 = n$) is presented, which requires a description of $O(\log\log m + n\sqrt{\log n})$ bits. [JvE86] reduce the upper bound for $O(1)$ time 1-probe (FKS-like) hashing to $O(\log\log m + n\log\log n)$-bits. [SvE84] give an $O(n)$-time 1-probe scheme and show that a variation of the FKS-scheme is space optimal at a cost of taking $O(n)$ time to hash a value. Their scheme turns out to be surprisingly similar to our $O(1)$ time 1-probe scheme, which is the first to be optimal in both space and time.

Non-oblivious $O(1)$-probe schemes turn out to be a lot more powerful than 1-probe schemes, where the question of obliviousness, of course, has no bearing. [FNSS88] present a non-oblivious scheme which needs only $O(\log\log m + \log n)$ bits, and [FN88] have recently shown that no additional memory is required when m is polynomial in n. No comparable oblivious schemes are known, and the natural question to resolve is how much of the exponential spatial advantage of $O(1)$-probe non-oblivious schemes over 1-probe schemes is due to their adaptive character, and how much is due to the opportunity to perform additional oblivious search. The counting arguments used in [Me84] seem to provide no help in an effort to construct lower bounds for multi-probe oblivious hashing: the $\Omega(n)$-bit portion of the argument collapses even for 3-probe schemes.

We show that the spatial complexity of k-probe oblivious hash functions for full tables is lower bounded by $\Omega(n\alpha^k + \log\log m)$, and that this bound is tight with $e^{-1-2\log k/k} \leq \alpha \leq e^{-1+5/k}$. In contrast to 1-probe schemes, no comparable lower bound can be obtained for $O(1)$-probe schemes for load factors less than 1. Indeed, a probabilistic construction shows that k-probe oblivious hash functions can be specified in as few as $O(\log n + \log\log m)$ bits, when the size n_1 of the hash table is $(1+\epsilon)n$, $\epsilon > 0$.

For completeness, we note that Mairson [Ma83], [Ma84] analyzed a number of related problems including binary search adapted to a page oriented hash scheme, where the cost to read a 2^k-record page is fixed, and the data is sorted on each page. He analyzed a scheme limited to reading one page, and supporting k rounds of binary search. While his scheme is not oblivious since it uses binary search, its spatial complexity is remarkably close to the spatial complexity of fully oblivious k-probe schemes, analyzed in this manuscript.

In Section 2, we prove that the spatial complexity of a k-probe oblivious hash function for a set of n elements from the universe $U = [1, m]$ is $\Omega(e^{-k}n/k^2)$. Section 3 shows that a random set of $O(2^{ne^{-k}} \log m)$ k-probe hash functions contains a perfect k-probe hash function for each n element subset of U with probability $(1 - o(1))$. The first part of Section 4 exhibits an $O(1)$-time 1-probe hash scheme which uses only $O(\log \log m + n)$ bits of external memory, and thereby shows that the lower bound for 1-probe schemes can be met for $O(1)$ time hash functions. The second part of Section 4 gives several explicit potentially useful constructions of oblivious multi-probe hash functions. Our explicit constructions use the ideas of universal hashing, in that we exhibit a family of functions having the property that, for any $S \subset U$, only $1/n$ of the functions are not valid multi-probe functions. We also extend the lower bounds from [Me82] to a wider class of families of universal hash functions, show that the bounds are formally tight, and give explicit constructions that almost match the lower bounds.

2. Lower bounds for oblivious search

Mehlhorn's $\Omega(n)$ bound for the size of a perfect hash function, [Me82], is based on the following counting argument: the number of functions must be at least the number of n-item subsets, which belong to an m-element universe U, divided by the maximum number of subsets which can be mapped one-to-one into $[0, n-1]$ by a single hash function: $count \geq \binom{m}{n}/(m/n)^n$. Taking the logarithm of this estimate gives the size bound for such hash functions. Unfortunately, this ratio decreases by a factor of k^n when k-probe maps are allowed. Formally, a hash function defines a bipartite graph on $B = U \times \{0, \ldots, n-1\}$. In the 1-probe case, each vertex in U has degree 1, and the count of the number of perfect matchings afforded by a single function is precisely the number of different subsets serviceable by the hash function. Once k probes are allowed, the degree of each vertex increases by a factor of k, and the obvious $(km/n)^n$ counts the number of possible matchings from all n-subsets of U, which overcounts the number of serviceable subsets by as much as a factor of k^n.

Consequently, we are obliged to model the k-probe scheme more carefully. We model a family of such hash functions as a set $\bar{H}$ of bipartite graphs on B, where each node of U has degree k, for each graph. $\bar{H}$ contains a k-probe perfect hash function for each n-element subset of U if and only if each n-element subset has a perfect matching for some $H \in \bar{H}$. The lower bound for describing such k-probe oblivious hash functions is thus $\log |\bar{H}|$.

To estimate $|\bar{H}|$, we choose an arbitrary $H \in \bar{H}$ and a randomly selected n-element set $V \subset U$, and compute an upper bound $\bar{p}$ for the probability that $H \in \bar{H}$ provides a perfect matching for V. It then follows that the number of n element subsets of U, for which H is perfect is at most $\binom{m}{n}\bar{p}$. Thus

$$|\bar{H}| \geq 1/\bar{p} \tag{1}$$

and members of $\bar{H}$ cannot be identified in fewer than $\log_2 \frac{1}{\bar{p}}$ bits.

By letting V be selected at random, we may imagine an honest intermediary who conveys, as answers to queries by our (partial) matching algorithm, precise (minimal amounts of) information about the items selected.

We will estimate the probability that a carefully selected subset $\tilde{\eta} \in [1, n]$ of n/ck items is covered by edges emanating from V. (The exact value of c will be specified later).

For $v \in U$, we define $Image(v) = \{i \in [1,n] \mid \{v,i\}$ is an edge in the graph $H\}$,

For $i \in [1,n]$ we define $Preimage(i) = \{v \in U \mid \{v,i\}$ is an edge in the graph $H\}$.

Let $P_i = |Preimage(i)|$. Let the tuple η comprise the elements $i \in [1,n]$, sorted in order of increasing P_i values.

By construction, the first items in η are not overly likely to be covered by the edges emanating from a randomly selected set V. The selection process for $\tilde{\eta}$ proceeds as follows:

0. Let $V_0 \leftarrow V$, $\eta_0 \leftarrow \{1,\ldots,n\}$.
1. For $i = 0$ to $n/ck - 1$ perform 2 through 5:
   2. Let ν_i be an element with minimum P_{ν_i} in η_i.
   3. Sequence through the elements of V_i and stop at first $v \in V_i$ which is in $Preimage(\nu_i)$. If no such v is found return "fail", *H is not perfect for V*.
   4. $V_{i+1} \leftarrow V_i - v$.
   5. Set $\eta_{i+1} \leftarrow \eta_i - Image(v)$.

      Note that the items in η_{i+1} do not have a preimage in $V_0 - V_i$ and $|\eta_{i+1}| \geq |\eta_0| - ik \geq n - n/c$.
6. $\tilde{\eta} \leftarrow \eta_i$.

If the above procedure fails, then there is no matching from V to $[1,n]$. (The converse is of course not true, but our aim is to upper bound the probability of a success.)

We have to estimate the probability that V contains an element in $Preimage(\nu_i)$, for $0 \leq i < n/ck$, at step 3. The query at step 3 is whether $v \in V_i$ is in $Preimage(\nu_i)$. Only at step 5 is $Image(v)$ actually revealed. Since $P_{\nu_0} \leq km/n$, the probability that the j^{th} element in V_0 is the first one to cover ν_0, is

$$Prob\{v_j \in Preimage(\nu_0) \mid v_1,\ldots v_{j-1} \notin Preimage(\nu_0)\} \leq \frac{km}{n(m-(j-1))} \leq \frac{k}{n(1-n/m)}.$$

The estimates of the probabilities of success for $i > 0$, in step 3 are slightly more delicate. In particular, the event: "$v^i_j \in Preimage(\nu_i) \mid v^i_1,\ldots v^i_{j-1} \notin Preimage(\nu_i)$", has to be conditioned by the additional information concerning v^i_j, acquired by previous queries. However, all we have learned is that v^i_j is not in the $Preimage$ of some of the elements $\nu_0,\nu_1,\ldots,\nu_{i-1}$, that $v^i_j \notin V - V_{i-1}$ and $v^i_j \notin \{v^i_1,\ldots v^i_{j-1}\}$. The probability that $v^i_j \in Preimage(\nu_i)$ is maximized if all the eliminated candidates are not in $Preimage(\nu_i)$. Since ν_ℓ (selected in step 2) is among the $k\ell+1$ smallest elements in η_0, it follows that $\sum_{\ell=0}^{i-1} P_{\nu_\ell} \leq ikm/n$; also $V - V_{i-1} \subset \bigcup_{\ell=0}^{i-1} Preimage(\nu_\ell)$, and therefore in the most extreme case, the conditional information can eliminate at most $(\sum_{l=0}^{i-1} P_{\nu_l} + j - 1 < m/c + n)$ elements of U as candidates for v^i_j. It follows that:

$$Prob\{v^i_j \in Preimage(\nu_i) \mid \text{all previous events}\} \leq \frac{P_{\nu_i}}{(m - m/c - n)},$$

and

$$\begin{aligned} Prob\{\nu_i \in Image(V_i)\} \mid \text{ all previous events}\} &\leq 1 - \left(1 - \frac{P_{\nu_i}}{m - m/c - n}\right)^{n-i} \\ &\leq 1 - (1 - o(1))e^{-P_{\nu_i}\frac{n}{m-m/c-n}}. \end{aligned}$$

And finally

$$\begin{aligned} p = Prob\{H \text{ is perfect for } V\} &\leq Prob\{ \text{ that the } Image \text{ of } V \text{ contains all the elements in } \tilde{\eta}\} \\ &\leq \prod_{i=0}^{n/ck-1} \left(1 - (1 - o(1))e^{-P_{\nu_i}\frac{n}{m-m/c-n}}\right). \end{aligned}$$

Since $\sum_{l=0}^{i-1} P_{\nu_l} \leq ikm/n$, by construction, the above product is maximized, effectively, when all P_{ν_i} are set to km/n. We choose $c = (k+1)m/(m-n)$ and conclude that

$$p = Prob\{H \text{ is perfect for } V\} \leq \left(1 - (1 - o(1)))e^{-(k+1)/(1-n/m)}\right)^{n(1-n/m)/(k+1)k}.$$

Computing $\log_2 \frac{1}{p}$ gives roughly $\left(\frac{n}{k^2}e^{-(k+1)/(1-n/m)}\right)$ bits necessary to specify an oblivious search strategy. We have proved

Theorem 1.

The spatial complexity of a k-probe oblivious hash function for a set of n elements belonging to the universe $[1, m]$ is $\Omega(\frac{n}{k^2}e^{-k\frac{m}{m-n}})$, (or $\Omega(\frac{n}{k^2}e^{-k})$ for $n = o(m)$). ∎

The above lower bound for $|\bar{H}|$ is not an increasing function of m.

However, a simple information theoretic argument, which was mentioned to us independently by [FN] and [F], shows that $\Omega(\log m/(k \log n))$ is also a lower bound on the size of $|\bar{H}|$. The argument is given in [FNSS88] and is reproduced here for completeness. A k-probe hash function H maps elements of U into $\{1, \ldots, n\}^k$. One of the conditions a k-probe scheme satisfies is that no $k+1$ items in S have all k probe locations in common. Thus

$$|\bar{H}| \geq \frac{\log(m/k)}{\log\binom{n}{k}}, \tag{2}$$

and $\Omega(\log\log(m) - k\log(n))$ bits are needed to name H.

Combining the information theoretic bound with Theorem 1 we get

Theorem 2.

The spatial complexity of a k-probe oblivious hash function for a set of n elements belonging to the universe $[1, m]$ is $\Omega(\log\log m + \frac{n}{k^2}e^{-k\frac{m}{m-n}})$. ∎

By appealing to a hypergraph model and extending the proof technique of Theorem 1, we can incorporate the method of Fredman and Komlos [FK84] to show that $|\bar{H}|$ is actually lower bounded by the product of the two bounds, rather than the maximum of the two. This gives the slightly stronger result

Theorem 3.

The size $|\bar{H}|$ of a family of k-probe hash functions from a universe U to a table of size n is: $\Omega((1 + \alpha^{k\frac{m}{m-n}})^{n/k^2}\frac{\log(m/k)}{\log(n^k)}$. ∎

The above theorems also show that the lower bound of $\Omega(n)$ for the spatial complexity holds even for relatively small universes (such as $m \approx 2n$). On the other hand, for $k > 1$, the lower bound does not hold if we were to map n elements into a table of size $(1+\epsilon)n$. The upper bounds in the second part of Section 3 show that this is not an artifact of our proof method.

3. Upper bounds for oblivious search

3.1 Oblivious k-probe schemes

Our lower bounds suggest that, while k-probe perfect hash schemes must have a reasonably large program length, in the case of oblivious search, the $k-1$ additional search opportunities might reduce the length by a factor of about e^k in its n dependence. We appeal to methods from the theory of random graphs to give a nonconstructive demonstration that this reduction is indeed possible, at least in a formal sense. The proof is actually a probabilistic construction of a k-probe oblivious hash function. For $k = 1$, such a probabilistic construction has been given in [Me82]. This proof for $k = 1$ is quite simple and, in fact, both the lower and upper bound were obtained by the same argument; when additional probes are permitted, the lower bound argument, as we have seen, is more delicate, while the upper bound, as we now demonstrate, requires much more care.

The procedure is to imagine constructing a random bipartite graph G on $U \times [1, n]$, where each vertex in U has degree k. We then select a random set S of n items from U and estimate the probability p that G contains a perfect matching on S. With positive probability, a family $\bar{H}$ of $\log_{1/(1-p)} \binom{m}{n}$ such random graphs will, for each n-element subset in U, contain some graph that has a perfect matching for that set, and it follows, therefore that such an $\bar{H}$ exists. The number of bits required to designate an element $H \in \bar{H}$ is then $\log \log_{1/(1-p)} \binom{m}{n}$.

Accordingly, let $S \in U$, $|S| = n$ be the subset we wish to match. Let, for convenience, G be used to name the portion of our random graph that is restricted to $S \times [1, n]$. A few preliminary remarks and definitions will help simplify the subsequent exposition.

- We recall Hall's (a.k.a. The Marriage) Theorem: A matching exists if and only if for all j, every set of j vertices in S has at least j vertices in $[1, n]$ that are connected to the set. Let Hall(j) be the property that every set of j vertices in S has at least j vertices in $[1, n]$ that are connected to the set.
- Let, for each $s \in S$, the first 3 random edges that are chosen to emanate from s be gold. The remaining $k-3$ edges will be green.
- Let A be the event that the Hall(j) property holds for $j \leq n/4$ for the subgraph restricted to the gold edges.
- Let B be the event that the Hall(j) property holds for $n/4 < j \leq n - n/(k-3)$, where both gold and green edges are used.
- We will be running a matching algorithm which will find augmenting paths via breadth first search on G, and will be selecting the random edges of G as the algorithm progresses. Let the set X comprise the first $n/4$ vertices of $[1, n]$ that are matched (by gold edges, as it happens).
- Let C be the event that the first $(k-4)n$ green edges encountered by the algorithm cover $[1, n] - X$. We could imagine an adversary taking note of matched vertices, which will be announced whenever the partial matching is augmented. It will also be told of every green edge that is encountered by the matching algorithm.

We use a standard matching algorithm (c.f. [HK73]): For each $s \in S$, a new breadth first search is initiated to find an augmenting path from s. A bfs level explores non matching edges from a current vertex set $R \subset S$, which is initially s. The search is successful for s successful if it discovers a vertex in $[1, n]$ that

has not been visited by any of the searches, and is therefore unmatched. Otherwise, the currently matched mates (in S) of the $[1,n]$ vertices that are newly encountered in the current level of the search (i.e., that have not been previously seen by the current search) are used to replace R for the next level of exploration. Once a previously unmatched vertex $\nu \in [1,n]$ is visited, the alternating property of being matched or not, along the path of edges from s to the ν is reversed. Then a new s is selected and a new search initiated.

Our use of the algorithm has two constructive phases, which are followed by a postamble. The first phase pursues a partial matching on the subgraph of gold edges, until a set X of $n/4$ vertices of S have been matched, or a subset of $j \leq n/4$ elements have been encountered that violates Hall(j). In the second phase, the count is extended, if possible, to n via both gold and green edges. If a match does not occur by the time $(k-4)n$ green edges have been used in phase 2, the algorithm fails. The postamble can be entered upon failure encountered during phases 1 or 2, or upon a successful matching (completed in phase 2). If success or failure occurs before $(k-4)n$ green edges are explored, we may imagine that the algorithm continues to select and report new green edges until the requisite number of edges is used.

The algorithm can switch to the fail state in phase 1 only if A does not hold. In phase 2, failure can be due to the fact that a new s does not have an augmenting path in a breadth first search that fails because A or B does not hold, or because the breadth first search touches $n(1-\frac{1}{k-3})$ vertices, which fail to have green edges that cover enough of $[1,n]-X$. In this later case, C does not hold.

These observations prove

Lemma 4.

$Pr\{\text{matching}\} \geq Pr\{A \cap B \cap C\}$. ∎

Consequently, $Pr\{\text{matching}\} \geq Pr\{B \cap C|A\}Pr\{A\} \geq (Pr\{B|A\} + Pr\{C|A\} - 1)Pr\{A\}$.

We now estimate these probabilities. The gold edges, for each vertex, are selected without replacement (i.e., each triple of edges comprises three distinct members), but the remaining $k-3$ edges are selected with replacement for computational convenience. It is also helpful to denote $[1,n]-X$ by $\{y_1, y_2, \ldots, y_{3n/4}\}$. There follows:

$$Pr\{A\} \geq 1 - \sum_{3<j\leq n/4} \binom{n}{j}\binom{n}{j-1}(\frac{j-1}{n}\frac{j-2}{n}\frac{j-3}{n})^j = 1 - O(n^{-4}).$$

$$Pr\{B|A\} \geq 1 - \sum_{n/4<j\leq n(k-4)/(k-3)} \binom{n}{j}\binom{n}{j-1}(\frac{j-1}{n})^{(k-3)j} > 1 - o((4/5)^{3n/4}),\ k > 6.$$

Let $hit(y)$ be the event that y is hit by one of the first $(k-4)n$ green edges examined. Note that $Pr\{C|A\} = Pr\{C\}$.

$$\begin{aligned} Pr\{C\} &= Pr\{hit(y_1)\} \times Pr\{hit(y_2) \mid hit(y_1)\} \times \cdots Pr\{hit(y_{3n/4}) \mid hit(y_1), \ldots, hit(y_{3n/4-1})\} \\ &\geq (1 - (\frac{n-1}{n})^{n(k-5)})^{3n/4} > (1 - e^{-k+5})^{3n/4}. \end{aligned}$$

Since $1 - Pr\{B \mid A\} = o(Pr\{C \mid A\})$, for $k > 6$, and $Pr\{A\} = 1 - o(1)$, the probability p of a matching is $\geq (1 - e^{-k+5})^{3n/4}(1 - o(1))$.

As we have already observed, a family of $|\bar{H}| = \log_{1/(1-p)}\binom{m}{n}$ such random graphs can supply a hash function (graph) for all n-element subsets. Computing $\log_2 \log_{1/(1-p)}\binom{m}{n}$ gives, $b \approx \log_2 \log_2 m + \frac{3}{4}e^{5-k}n$

bits to designate which random graph gives a k-probe perfect hash function for a given set S. If time and workspace are of no significance, we can follow the theme given for 1-probe hashing in [Me82], which is to use the lexicographically smallest collection $\bar{H}$ among all collections of such 2^b bipartite graphs which supports k-probe hashing for all n-item subsets of $[0, m-1]$. Then the b-bit number H names a graph within that collection (with respect to the lexicographic order within $\bar{H}$). The desired datum is found in at most k probes, as specified by the graph's edges.

We have shown

Theorem 5.

A k-probe hash function for a set of n elements belonging to the universe $[1, m]$ into a table of size n can be specified in $\log\log m + O(ne^{-k})$ bits, which is tight to within a factor of k^2 of the lower bound, (or tight to within a constant factor for constant k). ∎

Finally, we remark that a straightforward application of Hall's Theorem shows that if we were to map the elements of S into a table of size $2n$, then the probability of a successful matching using 3 probes is $1 - o(n^{-4})$. It follows that

Observation 6.

A 3-probe hash function for a set of n elements belonging to the universe $[1, m]$ into a table of size $2n$ can be specified in $O(\log\log m + \log n)$ bits. ∎

4. Explicit constructions and related bounds

4.1 An optimal $O(1)$-time 1-probe scheme

The upper bound in the previous section uses a probabilistic construction to establish the spatial complexity of k-probe oblivious hash functions. Unfortunately, such a technique does not yield any explicit $O(1)$-time hash function.

The hashing technique given in [FKS84] uses a table of linear congruential functions of the form $(ax \bmod p) \bmod q$. The basic idea, which we detail below, is to use such a hash function to define an implicit partition of the data set into a favorably distributed collection of collision buckets, and to look up, given a bucket index, an explicit (locally defined) secondary hash function which determines a unique address (within the bucket) for item sought.

Altogether, the address evaluation uses a few $\log n$-bit arithmetic computations, several array accesses, and one long word computation of the form $(kx \bmod p) \bmod n^2$, where $k, p \in U$ and $p < n^2 \log m$. It is natural to ask if the [FKS84] scheme or some other reasonable hash scheme can be can be expressed in an optimal $O(\log\log m + n)$ bits, while maintaining $O(1)$ time search, as conjectured in [SvE84].

We show that an [FKS84]-based scheme can be expressed in an optimal number of bits (up to a multiplicative factor), and performed with essentially the same selection of $O(1)$ arithmetic operations and array accesses.

We use the usual machine model associated with this problem, which is somewhat idealized. The model is basically that of a random access machine. In particular, an array access of an $O(\log n)$-bit word takes unit time, and index computations are permitted. Words can be added, subtracted, multiplied and

(integer) divided in constant time. We also use the same single long word computation as in [FKS84] and [Me82] to map elements from U into $[0, n^2]$. Similarly, we employ the expositional expediency of calling this computation an $O(1)$ time operation.

The original FKS construction has four basic steps.

1. A (long word) function $h_\alpha(x)$ is found that maps S into $[0, n^2 - 1]$ without collisions, so that all subsequent computation can use normal length words. $h_\alpha(x) = (kx \bmod p) \bmod n^2$, $k, p < n^2 \log m$ and p prime. (Corollary 2 and Lemma 2 in [FKS84].) Let Σ be the image of $h_\alpha \circ S$.
2. Next, a function $h_\beta(x)$ is found that maps Σ into $[0, n-1]$ so that the sum of the squares of the collision sizes is not too large. $h_\beta(x) = (\kappa x \bmod \rho) \bmod n$, $\rho > n^2$ is prime and $\kappa \in [0, \rho]$ so that $\sum_{0 \le j < n} |h_\beta^{-1}(j) \cap \Sigma|^2 < 3n$. (Corollary 3 in [FKS84].)
3. For each hash bucket (i.e., integer having at least one appearance in the multiset $\{h_\beta(t)\}_{t \in \Sigma}$) a secondary hash function h_i is found that is one-to-one on the collision set. Let c_i be the size of the collision set for bucket i: $c_i = |h_\beta^{-1}(i) \cap \Sigma|$. Then for $x \in h_\beta^{-1}(i) \cap \Sigma$, $h_i(x) = (k_i x \bmod \rho) \bmod c_i^2$, where $k_i \in [0, \rho - 1]$. The item $s \in S$, (which is represented by $t = h_\alpha(s)$ and located in hash bucket $i = h_\beta(t)$) is stored in location $C_i + h_i(h_\alpha(s))$, where $C_i = c_0^2 + c_1^2 + \ldots + c_{i-1}^2$. This locates all n items within a size $3n$ table.
4. Finally, the table is stored without vacant locations in a size n array $A[1..n]$, i.e. the array contains the items of S in the order of appearance in the table $3n$.

The composite hash function requires h_α, h_β, a table $K[0..n]$ storing the parameters k_i for the secondary functions h_i, a table $C[0..n]$ listing the C_i, and a compression table $D[1..3n]$, where $D[j]$ gives the index, for A, of the item (if any) that hashes to the value j by the function outlined in 1 through 3. As presented, the description of this perfect hash function requires $2 \log\log m + O(n \log n)$ bits.

In choosing the secondary hash functions h_i for each bucket, we appeal to [FKS84]. They point out that since their existence proofs are based on expected case analysis, at least one half of the numbers in $[0, \rho - 1]$ will yield appropriate functions h_i if the hash range is doubled. In particular, their Corollary 4 shows that given a collision set of any c_i items in Σ, at least half of the numbers in $[0, \rho - 1]$ will, if selected as multiplier k_i, yield a function $h_i(x) = (k_i x \bmod \rho) \bmod 2c_i^2$ that is one-to-one on the set. Consequently, if we have z collision buckets requiring multipliers, we may select a single multiplier that is one-to-one for $\lceil z/2 \rceil$ of the buckets, provided we double the size of the hash range in each case. Altogether, we need at most $\lfloor \log n \rfloor + 1$ different k_i values, where k_i is the i^{th} multiplier servicing about $1/2^i$ of the buckets. Moreover, the information content of the map from bucket indices into these $\lfloor \log n \rfloor + 1$ multipliers (i.e., the representation of the table K with Huffman coding) is $O(n)$.

In summary, the optimal perfect hash function is that described above, with two modifications:

1. The (uncompressed) hash value is $h_i(x) = ((k_i x \bmod \rho) \bmod 2c_i^2) + 2C_i$.
2. The multiplier values k are iteratively selected to satisfy the one-to-one requirement of the maximum number of (unsatisfied) h_i's.

It remains to show how to achieve a compact encoding of the content of the tables K, C, and D to require only $O(n)$ bits, that is readily decodable in $O(1)$ time in our computational model. The basic decoding operations we will use are as follows:

1. Extract a subsequence of bits from one word.
2. Concatenate two bit strings of altogether $O(\log n)$ bits.

3. Locate the bit-index of the k-th zero in a word.
4. Count the number of consecutive 1-s in an $O(\log n)$-bit word, starting at the beginning.
5. Count the number of zeros in an $O(\log n)$-bit word.
6. Access a few constants.

The first two operations are a matter of arithmetic. The last four can be performed with small look-up tables. In particular, it suffices to break the words into words of $\frac{\log n}{2}$ bits (padded with leading zeros), and to use these words to access decoder arrays. The third operation, for example, requires for each k, $0 < k \leq \log n/2$, an array of $\sqrt{n}$ words. A word of $2\log n$ bits requires up to four accesses to compute the location of a k-th 0. The fourth and fifth operation is accomplished in a similar matter.

The table C, which contains the values c_i^2, is encoded as follows. First, the squares of the various lengths c_i^2 are stored in a table T_0 in unary notation (in order of appearance, separated by 0's). (Note that the bit length of T_0 is at most $4n$). We let str_i denote the string that encodes c_i^2. Because of operations 1 and 2, we may suppose that each bit of T_0 is addressable. Let a_i be the address (index) of the starting point of $str_{i\log n}$ in T_0, so that $0 \leq a_i < 4n$, for $i = 0, 1, \ldots, n/\log n$. A second table T_1 contains, in binary, the indices $a_0, a_1, \ldots, a_{n/\log n}$. If $a_{i+1} - a_i < 2\log n$, the table information for intermediate c_j^2 (i.e. for $i\log n \leq j < (i+1)\log n$) can be readily decoded via $O(1)$ accesses to T_0 and T_1 and $O(1)$ of our decoding operations.

The case $a_{i+1} - a_i \geq (\log n)^2$ is handled via a third table T_2, which stores, starting in (bit) location a_i, the $\log n - 1$ binary indices for the starting locations of str_j, $i\log n < j < (i+1)\log n$.

The remaining case, $2\log n \leq a_{i+1} - a_i < (\log n)^2$ is handled with an additional level of refinement. Let $b_{i,j}$ be the address (index in T_0) of the starting point of $str_{i\log n + j\log\log n}$, ($j < \log n/\log\log n$ and $a_i < b_{i,j} < a_{i+1}$). In T_2, we store, starting in location a_i, the binary offsets $b_{i,1} - a_i, b_{i,2} - a_i, \ldots$. Each offset is stored as a $2\lceil\log\log n\rceil$-bit binary number. If $b_{i,j+1} - b_{i,j} \leq 2\log n$ the information for intermediate c_l, $i\log n + j\log\log n < l < i\log n + j\log\log n$, is readily decoded; for all other cases, the offsets (of size $\log\log n$) of all intermediate $c_l's$ are encoded through one last level of indirection in a table T_3, starting at location $b_{i,j}$.

The table K can be encoded in exactly the same way. In particular, a table K_0 contains, for its i-th sequence of bits, the integer α_i in unary, if k_{α_i} is the multiplier assigned to hash bucket i, $0 \leq i < n$. (Recall that the first multiplier (encoded by the string "1") is usable for at least half of the hash buckets.) The total length of the sequence comprises at most n 0's, $n/2$ singleton 1's $n/4$, doubleton 1's, etc., for a total length of $3n$. The multipliers $k_1, \ldots, k_{\log n}$ are stored in a $\log n$-word array.

It is evident that we have encoded a perfect hash function in $O(n)$ bits. To summarize, the above construction, combined with the lower bound of [Me84], gives

Theorem 7.

The spatial complexity of a $O(1)$-**time** perfect (1-probe) hash function for a set of n elements belonging to the universe $[1, m]$ is $\Theta(\log\log m + n)$. ∎

4.2 Related constructions and corresponding bounds

We have analyzed the complexity of hash function families with the property that they contain, for any n-element $S \subset U$, at least one "good" hash function. Closely related are universal hash function families [CW79], which have the property that, for any such subset, **most** functions behave "adequately" well. It turns out that the existence of a good function in the first model is frequently proven via an averaging argument, which can usually be modified to yield a family of universal hash functions. Not surprising, the same type of functions are generally used in both models.

The appropriate definitions in [CW79] and [Me82] can be combined to define a family H of c-universal$_k$ hash functions for a universe U as follows.

Let H be a collection of functions: $U \mapsto [1,n]$. We call H c-universal$_k$ if[1]:

$$\text{for } j \le k,\ x_1 < x_2 < \cdots < x_j \in U:\ Prob_{h \in H}\{h(x_1) = h(x_2) = \cdots = h(x_j)\} < \frac{c}{n^{j-1}}.$$

One can also define a family of universal hash functions with respect to any property Q and probability p: We say that H is $(p;Q)$-universal if for any suitably restricted $S \subset U$,

$$Prob_{h \in H}\{h \text{ restricted to } S \text{ has property } Q\} > p.$$

We first derive upper and lower bounds for c-universal$_k$ hash functions and then show how to construct universal families of oblivious $\log n$-probe hash functions.

A useful lemma, for establishing universal properties was established in [CW79] and extended in [WC79]:

Lemma [CW79].

Let $p > m$ be a prime. Let $H = \{h \mid h \text{ is a polynomial of degree } k-1 \text{ over the field } Z_p\}$. Let $F = \{f \mid f = h \bmod n, h \in H\}$. The family F of functions: $U \mapsto [1,n]$ is $(1+o(1))$-universal$_k$. ∎

Another useful lemma was established in [KU86], a variation of which we reproduce here:

Lemma [KU86].

Let p_m be a fixed prime greater than m.
Let $H_\xi = \{h \mid h(x) = g(x) \bmod (n/c),\ g \text{ polynomial of degree } \xi \text{ over the field } Z_{p_m}\}$.
Then for $j > \max(\xi, c),\ S \in U,\ |S| = n:\quad Prob_{h \in H_\xi}\{\max_{1 \le i \le n/c} |h^{-1}(i) \cap S| \ge j\} < n(\frac{c}{j-\xi})^\xi$. ∎

4.2.1 C-universal$_k$ hashing

Mehlhorn ([Me82]) showed that c-universal$_2$ hash functions have a spatial complexity of $\Omega(\log n + \log\log m - \log c)$ bits, and gave an explicit construction of a family of 8-universal$_2$ hash functions of matching spatial complexity. The following slight variation of his proof shows that a family of c-universal$_k$ hash functions must contain at least $\frac{n^k}{c}(\frac{\log(m/k)}{\log n} - 1)$ functions, and the spatial complexity of a function in the set is therefore $\Omega(k\log n + \log\log m - \log c)$ bits.

[1] All random selection will be with respect to the uniform distribution on the set in question.

Let $H = \{h_1, \ldots h_{|H|}\}$ be a c-universal$_k$ family of hash functions. Since the range of h_i is $\{1, \ldots, n\}$, there is a set $S_1 \subset U$, $|S_1| \geq m/n$, where h_1 is constant when restricted to S_1. By induction, there are sets $S_j \subset S_{j-1} \subset \cdots \subset U$: $|S_j| \geq m/n^j$, and $h_1, h_2, \ldots h_j$ are constant when restricted to S_j. Let j_0 be the maximal j for which $|S_j| \geq k$. Let $x_1, \ldots x_k \in S_{j_0}$. Since H is c-universal$_k$, and all x_i collide under $h_1, \ldots, h_{j_0}$, we must have $|H| > \frac{n^{k-1}}{c} j_0$. Since $j_0 > \frac{\log(m/k)}{\log n} - 1$, $|H| > \frac{n^{k-1}}{c}(\frac{\log(m/k)}{\log n} - 1)$, and hence $\log|H| > (1 - o(1))(k - 1\log n - \log c + \log\log m)$.

On the other hand, one can easily show that a randomly selected set of $\frac{1}{c} n^{k-1} \log \binom{m}{k}$ hash functions, (for $c \geq 3$), is c-universal$_k$ for U.

Remark 8: The spatial complexity for c-universal$_k$ hash functions, (for $c \geq 3$), is $(1 \pm o(1))(\log\log m + k\log n - \log c)$.

It is also not difficult to construct an explicit family of $(1 + o(1))$-universal$_k$ hash functions that is quite small, albeit not optimal. We first appeal to the following variation of Lemma 2 from [FKS84]:

Remark 9: Let $\Re_j = \{p \mid p \text{ prime and } p \in (n^j \log m, (2+\epsilon) n^j \log m)\}$. Then

$$\forall x \neq y \in U : \; Prob_{p \in \Re_j}\{x = y \bmod p\} < n^{-j}$$

Proof: By the prime number theorem we have

$$\sum_{p \in \Re_j} \log p = (1+\epsilon) n^j \log m + o(n^j \log m).$$

Let $P_{x,y} = \{p \mid p \text{ prime and divides } x - y\}$. Then

$$\sum_{p \in P_{x,y}} \log p \leq \log m,$$

and it follows that $|P_{x,y} \cap \Re_j| / |\Re_j| < n^{-j}$. ∎

Let $H_1 = \{h \mid h(x) = (ax + b \bmod p) \bmod n^k, p \in \Re_k, \text{ and } a \neq 0, b \in \{0, 1, 2, \ldots p-1\}\}$.
Let p_k be a fixed prime greater than n^k.
Let $H_2 = \{h \mid h \text{ is a polynomial of degree } k-1 \text{ with coefficients in the field } Z_{p_k}\}$.
Put $F = \{f \mid f(x) = (h_2 \circ h_1(x) \bmod p_k),\ h_2 \in H_2,\ h_1 \in H_1\}$. It follows that for $j \leq k$, $x_1 < x_2 < \ldots < x_j \in U$,

$$Prob_{f \in F}\{f(x_1) = f(x_2) = \cdots = f(x_j)\} < j^2 n^{-k} + (1 + o(1)) n^{-j+1},$$

and F is $(1 + o(1))$-universal$_k$ for U with spatial complexity $k(k+3)\log n + 3\log\log m$.

4.2.2 Hash functions with large probe numbers

In this subsection, we give a construction for k-probe hash functions of interest. Our probabilistic upper bounds from Section 3 show that there is a $\log n$-probe oblivious search strategy for full tables, and a 3-probe oblivious search strategy for 50% full tables, which each use only $O(\log\log m + \log n)$ additional space. We describe an explicit construction of a class of uniform oblivious $\log n$ probe hashing schemes, which map elements from a set S of size n into a table of size $(1+\epsilon)n$, $(\epsilon > 0)$, and which can be described in $O(\log\log m + \log^2 n)$ bits.

Construction 10.

We construct a family $\vec{H} = \{< f, f_1, \ldots, f_{k-1} >\}, k = O(\log n)$, of functions from U into $[1, \ldots (1+\epsilon)n]$. $\vec{H}$ will have the property that for any n-element set $S \subset U$, a randomly chosen $h \in \vec{H}$ will, with high probability, be a valid k-probe hash function mapping S into $[1, \ldots (1+\epsilon)n]$.

Let R_1 be the set of all primes in $(n^3 \log m, (2+\epsilon)n^3 \log m)$.

Let $G_1 = \{h \mid h(x) = (ax+b \pmod p) \pmod{n^3}, p \in R_1, \text{ and } a \neq 0, b \in \{0, 1, \ldots p-1\}\}$.

Let p_3 be a fixed prime greater than n^3, and set $c_2 = 2 + 4/\epsilon$.

Let $G_2' = \{h \mid h \text{ is a polynomial of degree } c_2 \log n \text{ over the field } Z_{p_3}\}$.

Let $G_2 = \{h \mid h = h' \bmod \left(\frac{n}{c_1 \log n}\right), h' \in G_2'\}$, where $c_1 = 2c_2/\epsilon$.

Finally, put $k = ((1+\epsilon)c_1) \log n$, and let

$$\vec{H} = \{< f, f_1 \ldots, f_{k-1} > \mid f = h_2 \circ h_1, (h_1, h_2) \in G_1 \times G_2 \text{ and } f_j(x) = f(x) + jn/(c_1 \log n)\}.$$

The family $\vec{H}$ of k-probe functions is *universal* in the following sense:

$$Prob_{h \in H}\{h \text{ is a valid } k\text{-probe function for } S\} \geq 1 - 2n^{-1}$$

Proof: The family G_1 is *universal* with respect to the property Q, and probability $\left(1 - n^{-1}\right)$: $Q = \{\forall x, y \in S \quad h(x) \neq h(y)\}$, with probability $1 - 1/n$. That is, for a randomly chosen $h_1 \in G_1$: $Prob\{\exists x, y \in S \text{ s.t. } h(x) = h(y)\} < 1/n$ (This follows from Remark 9.)

The family G_2 is *universal* with respect to the property Q_1, and probability $\left(1 - n^{-1}\right)$: $Q_1 = \{\forall i \in \{1, \ldots, n/(c_1 \log n)\} | g^{-1}(i)| \leq (1+\epsilon)c_1 \log n\}$. That is, for a randomly chosen $h_2 \in G_2$, restricted to S: $Prob\{Q_1\} \geq \left(1 - n^{-1}\right)$. (This follows from Lemma[KU].)

Let $f = h_2 \circ h_1$ be a randomly chosen function in $\vec{H}$. By construction, h_1 is collision free on S with probability $1 - 1/n$ and the collison chains created by h_2 are bounded by $(1+\epsilon)c_1 \log n$ with probability $1 - 1/n$. The k-probe function $< f, f_1 \ldots, f_{k-1} >$, as defined above, is therefore a valid $O(\log n)$-probe hash function for S with probability $1 - (2/n)$. ∎

The bit-complexity of the k-probe function is the same as for the function f, which is $O(\log \log m + \log^2 n)$.

Conclusions and open problems

We have shown how to construct explicit space-time optimal perfect hash functions, and given some related universal hashing constructions. We have given tight bounds on the spatial complexity of oblivious k-probe hash functions, and helped quantify the difference between oblivious and non-oblivious strategies. In particular, we have shown that, for full tables, $O(1)$ addtional oblivious probes can reduce the requisite size of the search program by a constant factor, but no more. As is well known, the decrease in program size, for partially full search tables, is more dramatic: probabilistic arguments show the formal existance of oblivious $O(1)$ probe schemes that need only $O(\log \log m + \log n)$ bits. It would be interesting to find an explicit construction of a small $O(1)$-probe oblivious family of hash functions that map n elements into a table of size $(1+\epsilon)n$.

Acknowledgements

The authors thank A. Fiat, M. Fredman, M. Naor, J. Spencer and P. van Emde Boas for helpful discussions.

References

[A85] M. Ajtai. "A Lower Bound for Finding Predecessors in Yao's Cell Probe Model," IBM RJ 4867 (51297) 10/3/85, Computer Science.

[AFK83] M. Ajtai, M. Fredman and J. Komlós. "Hash Functions for Priority Queues," 24*th* Ann. Symposium on Foundations of Computer Science, 1983, pp. 299–303.

[BBDOP86] F. Berman, M.E. Bock, E. Dittert, M.J. O'Donnell and D. Plank. "Collections of Functions for Perfect Hashing," SIAM Journal on Computing, Vol 15, No. 2, May 1986, pp. 604–618. July 1984, pp. 538–544.

[CW79] J. L. Carter and M. N. Wegman "Universal Classes of Hash Functions", Journal of Computer and System Sciences 18, 143-154 (1979).

[FN] A. Fiat and M. Naor. Personal communication, 1988.

[FN88] A. Fiat and M. Naor. "Implicit Hashing," manuscript.

[FNSS88] A. Fiat, M. Naor, J. P. Schmidt and A. Siegel, "Non-Oblivious Hashing," Proc. 20*th* ACM Symp. on Theory of Computing, 1988.

[F] M.L. Fredman. Personal communication, 1988.

[FK84] M.L. Fredman and J. Komlós. "On The Size Of Separating Systems And Families Of Perfect Hash Functions," SIAM Journal of Algebraic and Discrete Methods, Vol 5, No. 1, March 1984, pp. 61–68.

[FKS84] 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, Vol 31, No. 3, July 1984, pp. 538–544.

[G81] G.H. Gonnet. "Expected Length of the Longest Probe Sequence in Hash Code Searching," Journal of the Association for Computing Machinery, Vol 28, No. 2, April 1981, pp. 289–304.

[GM79] G.H. Gonnet and J. Ian Munro. "Efficient Ordering of Hash Tables," SIAM Journal on Computing, Vol 8, No. 3, August 1979, pp. 463–478.

[HK73] J.E. Hopcroft and R.M. Karp "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs," SIAM Journal on Computing, Vol 2, No. 4, (1973), pp. 225–231.

[JvE86] C.T.M. Jacobs and P. van Emde Boas. "Two Results on Tables," Information Processing Letters 22 (1986), pp. 43–48.

[KU86] A. Karlin and E. Upfal "Parallel Hashing - an Efficient Implementation of Shared Memory," 18th Annual symposium on Theory of Computing, May, 1986, pp. 160–168.

[Ma83] H. G. Mairson. "The Program Complexity of Searching a Table," 24th Symposium on Foundation of Computer Science, November, 1983.

[Ma84] H. G. Mairson. "The Program Complexity of Searching a Table," PhD Dissertation, Stanford, 1984, STAN-CS-83-988.

[Me82] K. Mehlhorn. "On the Program size of Perfect and Universal Hash functions," Proc. 23*rd* Ann. Symp. on Foundations of Computer Science, 1982, pp. 170–175.

[Me84] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, Springer-Verlag, Berlin Heidelberg, 1984.

[SS80] J.P. Schmidt and E. Shamir. "An Improved Program for Constructing Open Hash Tables," ICALP 80, Proceedings, pp. 569–581.

[SvE84] C. Slot and P. van Emde Boas "On Tape versus Core; An Application of Space Efficient Hash Functions to the Invariance of Space", 16th Annual symposium on Theory of Computing, May, 1984.

[TY79] R.E. Tarjan and A.C. Yao. "Storing a Sparse Table," Communications of the ACM, Vol 22, No. 11, November 1978, pp. 606–611.

[WC79] M. N. Wegman and J. L. Carter "New Classes and Applications of Hash Functions", 20*th* Annual Symposium on Foundations of Computer Science, October 1979, pp. 175–182.

[Y81] A.C. Yao. "Should Tables Be Sorted?," Journal of the Association for Computing Machinery, Vol 28, No. 3, July 1981, pp. 615–628.