

Guided tour of some results on hashing and dictionaries

by Rasmus Pagh (pagh@brics.dk)

February 26, 2001

This document presents some results on dictionaries in the form of step-by-step problems. Familiarity with the material on universal hash functions in the first four sections of [3] is assumed. Also, some basic probability theory is needed (see e.g. [4]). Emphasis is on simple ways of doing things, and some results go further back than the cited references. The last few parts of several problems are difficult (marked with * or **), but these can easily be skipped. Answers can be found in the cited references (though sometimes in a more general form than asked for here).

1 Perfect hashing

A *perfect hash function* for a set of keys S is a function that maps no two elements of S to the same value (is 1-1 on S). Such a function with range of size $O(|S|)$ can be used to solve the static dictionary problem in linear space, if the description of the hash function itself uses linear space. This problem looks at a simple design of such a function proposed in [5].

Let $f : U \rightarrow \{1, \dots, |S|\}$ and $g : U \rightarrow \{1, \dots, 10|S|\}$ be chosen independently and uniformly at random from “nearly universal” families of functions. Our candidate for a perfect hash function is $h(x) = (f(x) + a_{g(x)}) \bmod |S|$, where $a_i \in \{1, \dots, |S|\}$ must be suitably chosen for $i = 1, \dots, 10|S|$. Let $b_i = |\{x \in S \mid g(x) = i\}|$, and let σ be a permutation on $\{1, \dots, 10|S|\}$ such that $b_{\sigma(j)} \geq b_{\sigma(j+1)}$ for $1 \leq j < 10|S|$.

1. Show that the expected value of $\sum_i \binom{b_i}{2}$ is $< |S|/5$ (Hint: Rewrite the sum to a sum over all pairs in S).
2. Show that with probability $\geq 1/5$, $\sum_i \binom{b_i}{2} < |S|/4$. *From now on we assume that g has been found such that this is true.*
3. Show that with probability $\geq 1/2$ no two elements in S have the same function values under both f and g (Hint: The expected number of such pairs is $< 1/2$). *From now on we assume that f has been found such that this is true.*
4. Suppose that values of $a_{\sigma(1)}, \dots, a_{\sigma(j)}$ have been fixed, and that $b_{\sigma(j+1)} > 1$. Looking only at the set $\{x \in S \mid h(x) \in \{\sigma(1), \dots, \sigma(j+1)\}\}$, show that there are at most $4 \sum_{i=1}^j \binom{b_i}{2}$ “bad” values of $a_{\sigma(j+1)}$ (resulting in collisions for h).
5. Show that suitable values of the a_i can be determined in the order $i = \sigma(1), \sigma(2), \dots$.
6. (*) Show how to construct the perfect hash function in expected linear time (you may assume the result of problem 4, i.e., that random f and g can be picked in constant time).

2 A dynamic dictionary

Problem 1 showed that a static set can be stored with worst case constant lookup time. But what if one wants to perform insertions and deletions? This problem looks at such a dynamic dictionary, proposed in [6]. For simplicity, we start by assuming that we have access to truly random functions $f, g : U \rightarrow \{1, \dots, r\}$, where r is a parameter satisfying $2|S| \leq r \leq 5|S|$.

The data structure has two hash tables T1 and T2, and any key $x \in S$ is stored in either T1[f(x)] or T2[g(x)]. This makes lookup and deletion very easy. The insertion algorithm (for x) uses the “cuckoo approach”, by simply inserting the element in T1[f(x)], thus forcing any element in this cell to be inserted in T2, and so on, until an empty spot is found:

```
insert-x:
  if T1[f(x)]==empty then T1[f(x)]=x; goto end;
  x <-> T1[f(x)];
  if T2[g(x)]==empty then T2[g(x)]=x; goto end;
  x <-> T2[g(x)];
  goto insert-x;
end:
```

1. Show that the elements of S can be placed “correctly” in the two tables (i.e., in the cells given by f and g) if and only if there is no subset $S' \subseteq S$ such that $|f(S')| + |g(S')| < |S'|$ (Hint: Use Hall’s theorem on bipartite matchings).
2. Show that the probability that the elements cannot be placed correctly is $O(1/|S|)$ at each insertion (Hint: Sum the probability over all possibilities for S' , $f(S')$ and $g(S')$).
3. Show that insertion terminates if the new set can be placed correctly in the two tables.
4. Show that each insertion that terminates takes expected constant time.
5. (*) Show how to rebuild the datastructure when the above insertion algorithm does not terminate or the requirement on r is violated. The time should be expected amortized constant.
6. (**) Show that it suffices for f and g to be k -wise independent for $k = O(\log n)$ (i.e., the function values of any k elements are uniform and independent when choosing a random function).

3 Reliable hash functions

In sections 3 and 4 of [3], a constant expected time bound per operation of chained (universal) hashing was exhibited. This problem considers the possibility of making the constant time bound hold with high probability, as described in [2].

1. Argue that an expected time bound may leave a fair risk of performing considerably worse than expected.
2. Following [3, Section 3], show $E[\sum_i T(OP_i(k_i))] \leq 3m$ and $\text{Var}(\sum_i T(OP_i(k_i))) \leq m$.

3. Use Chebychev's inequality to strengthen the analysis of [3, Section 3] to yield

$$\Pr\left[\sum_i T(\text{OP}_i(k_i)) > 4m\right] < 1/m$$

4. In analogy with [3, Section 4], show that the above analysis goes through if the hash values of any four elements are independent, i.e., for all distinct $x_1, x_2, x_3, x_4 \in U$ and all $y_1, y_2, y_3, y_4 \in \{1, \dots, N\}$ we have $\Pr[h(x_i) = y_i, \text{ for } i = 1, 2, 3, 4] \leq 1/N^4$.

5. (*) Show that the family containing the functions $h_{\bar{a}}(x) = (a_3x^3 + a_2x^2 + a_1x + a_0 \bmod P) \bmod N$ for all $1 \leq a_0, a_1, a_2, a_3 < P$ has “nearly” the property needed in 4, i.e., the inequality holds with constant 2 rather than 1. (Hint: The 4 by 4 “Vandermonde” matrix M with $M_{ij} = x_i^j$ is invertible in $GF(P)$).

6. Show that the above family can be used to achieve a guarantee similar to that of 3.

4 An efficient universal family

This problem looks at a universal hash family, proposed in [1], that is computationally less expensive than $(ax \bmod P) \bmod N$, and also avoids the need to find a prime of the right size. The family consists of the functions from $\{0, 1\}^w$ to $\{0, 1\}^r$ of the form $h_a(x) = (ax \bmod 2^{w+r}) \text{ div } 2^w$, where $a \in \{0, 1\}^{w+r}$.

1. Explain how the above function can be implemented more efficiently than $(ax \bmod P) \bmod N$.
2. Take $0 \leq x < 2^w$ and let k be the largest integer such that 2^k divides x . Show that the multiset $\{ax \bmod 2^{w+r} \mid a \in \{0, 1\}^{w+r}\}$ consists of 2^{w+r-k} copies of each multiple of 2^k .
3. Show that the above family is universal.

References

- [1] Martin Dietzfelbinger. Universal hashing and k -wise independent random variables via integer arithmetic without primes. In *Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science (STACS '96)*, pages 569–580. Springer-Verlag, Berlin, 1996.
- [2] Martin Dietzfelbinger, Joseph Gil, Yossi Matias, and Nicholas Pippenger. Polynomial hash functions are reliable (extended abstract). In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, volume 623 of *Lecture Notes in Computer Science*, pages 235–246. Springer-Verlag, Berlin, 1992.
- [3] Peter Bro Miltersen. Universal Hashing. Lecture note, 12 pp, 1998.
- [4] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.
- [5] Rasmus Pagh. Hash and Displace: Efficient Evaluation of Minimal Perfect Hash Functions. In *Proceedings of the 6th international Workshop on Algorithms and Data Structures (WADS '99)*, volume 1663 of *Lecture Notes in Computer Science*, pages 49–54. Springer-Verlag, Berlin, 1999.
- [6] Rasmus Pagh and Flemming Friche Rodler. Dictionaries and perfect hashing: New algorithms and experiments. In preparation, 2001.