Perfect Hashing

1 Introduction

Let U be a universe of size m, and let S be a subset of U of size n such that $n \le m$. Assume that S is static. The problem is to store S optimally (linear space) in a memory T such that the membership queries are also efficient (constant query time). The membership queries are of the form "Is $q \in S$, and where can it be found in T?" Fredman, Komlós and Szemerédi (1984) [1] described a data structure based on hashing scheme that uses $n + o(n)$ space for storing S such that membership queries take $\mathcal{O}(1)$ time. We discuss their construction in this note.

Previous constructions and the gap

A naive way is to store S in a sorted array of length n. But the query time is $\mathcal{O}(\log_2 n)$ by binary search.

- Tarjan and Yao (1979) show that $\mathcal{O}(n)$ space and worst case query time $\mathcal{O}(\log_n m)$ can generally be attained.
  - This means worst case query time is $\mathcal{O}(1)$ if m bounded by a polynomial in n (e.g. $m = \mathcal{O}(n^c)$ for some constant $c > 0$).
- Yao (1981) shows that m grows at least exponentially in n (e.g $m = \Omega(e^{2n})$), then $n + 1$ space and worst case query time 2 is attained.
- Yao (1981) points out that for the immediate range (e.g. $m = 2^{\sqrt{n}}$), the possibility of linear space and constant query time is not yet settled. Fredman, Komlós and Szemerédi (1984) settle this gap.

In the second section, we discuss a data structure achieving linear space and constant worst case query time for all m and n, in particular, we first construct a data structure in time $\mathcal{O}(mn)$ such that it uses space at most $6n$, and requires 5 queries. Then, we show that the worst case construction time can also be improved to $\mathcal{O}(n^3 \log m)$. It can also be made $\mathcal{O}(n)$ in expectation. In the third section, we refine the space to $n + o(n)$ retaining constant query time in the same construction time. Real RAM model is assumed, i.e., addition, subtraction, multiplication and division operations can be done in constant time.

2 Preliminary data structure

For simplicity, to store a set S, let the universe $U = \{1, 2, \ldots, m\}$ be such that $p = m+1$ for some prime p.[1] This is so that the set $\{0\} \cup U$ is the finite field $\mathbb{F}_p$. The notation $a \bmod b$ is used to denote the integer x, $x \in \{1, 2, \ldots, b\}$ such that $x \equiv a \bmod b$.

Given $W \subseteq U$ with $|W| = r$, $k \in U$ and $s \geq r$, let $h_{k,s} : U \to [s]$ be a hash function such that

$$h_{k,s}(x) = (kx \bmod p) \bmod s,$$

and for a given $1 \leq j \leq s$, let

$$\begin{aligned} B(s, W, k, j) &= |\{x \mid x \in W \text{ and } h_{k,s}(x) = j\}| \\ &= |W \cap h_{k,s}^{-1}(j)|, \end{aligned}$$

in words, $B(s, W, k, j)$ is the number of times the value j is attained by $h_{k,s}$ when restricted to W. Clearly, $\sum_{j=1}^{s} B(s, W, k, j) = |W| = r$.

Lemma 1. *1. There exists a $k \in U$ such that $\sum_{j=1}^{s} \binom{B(s,W,k,j)}{2} < \frac{r^2}{s}$.*

2. For at least half of the values of $k \in U$, we have $\sum_{j=1}^{s} \binom{B(s,W,k,j)}{2} < \frac{2r^2}{s}$.

Proof. We show that $\frac{1}{p-1} \sum_{k=1}^{p-1} \left[\sum_{j=1}^{s} \binom{B(s,W,k,j)}{2} \right] < \frac{r^2}{s}$. Then, by expectation argument, the proof follows. By definition,

$$\begin{aligned} \binom{B(s,W,k,j)}{2} &= |\{(x,y) \in W \times W \mid x < y \text{ and } h_{k,s}(x) = h_{k,s}(y) = j\}| \\ \implies \sum_{j=1}^{s} \binom{B(s,W,k,j)}{2} &= |\{(x,y) \in W \times W \mid x < y \text{ and } h_{k,s}(x) = h_{k,s}(y)\}| \\ \implies \sum_{k=1}^{p-1} \sum_{j=1}^{s} \binom{B(s,W,k,j)}{2} &= |\{(k,x,y) \in U \times W \times W \mid x < y \text{ and } h_{k,s}(x) = h_{k,s}(y)\}|. \end{aligned}$$

Now for fixed $(x, y) \in W \times W$ with $x < y$, the quantity $h_{k,s}(x) = h_{k,s}(y)$ is equivalent to

$$k(x-y) \bmod p \in \{s, 2s, \ldots, ls\} \cup \{-s, -2s, \ldots, -ls\},$$

where $ls < p$, but since p is a prime, $l \leq \lfloor (p-1)/s \rfloor$. Also, due to field properties of $\{0\} \cup U$ and $x < y$, $(x - y) \in U$ and it has a unique multiplicative inverse modulo p. Therefore, the number of such k's is at most $\frac{2(p-1)}{s}$. Now the number of pairs $(x, y) \in W \times W$ such that $x < y$ is $\binom{|W|}{2} = \binom{r}{2} < \frac{r^2}{2}$, and so

$$\frac{1}{p-1} \sum_{k=1}^{p-1} \sum_{j=1}^{s} \binom{B(s,W,k,j)}{2} < \frac{1}{p-1} \frac{2(p-1)}{s} \frac{r^2}{2} = \frac{r^2}{s}.$$

[1] Otherwise, find the next prime since we know at least one exists within a factor of 2 by Bertrand's postulate.

This completes proof of the first statement.

For the second statement, let X be a random variable that takes value $\sum_{j=1}^{s} \binom{B(s,W,k,j)}{2}$ for each $k \in U$ with uniform probability. By Lemma 1.1., $\mu = \mathbb{E}(X) < \frac{r^2}{s}$. The second statement can also be proved in two ways.

- By Markov, $\Pr[X \geq 2\mu] \leq \frac{1}{2}$, which implies, $\Pr[X < 2\mu < \frac{2r^2}{s}] > \frac{1}{2}$.
- Suppose on the contrary that for at least half of the values of $k \in U$, we have $X(k) \geq \frac{2r^2}{s}$. Then, $\sum_{k \in U} X(k) \geq \frac{p-1}{2} \frac{2r^2}{s}$, and so $\mu \geq \frac{r^2}{s}$. This is a contradiction.

□

The next two corollaries will especially be helpful in the construction of the data structure.

Corollary 2. *1. There exists a $k \in U$ such that the function $h_{k,r}$ partitions W into r blocks and the sum of squares of their sizes is strictly less than $3r$, i.e., $\sum_{j=1}^{r} B(r,W,k,j)^2 < 3r$.*

2. For at least half of the values $k \in U$, the function $h_{k,r}$ partitions W into r blocks such that the sum of squares of their sizes is strictly less than $5r$, i.e., $\sum_{j=1}^{r} B(r,W,k,j)^2 < 5r$.

Proof. First, observe that $\sum_{j=1}^{r} B(r,W,k,j) = \sum_{j=1}^{r} |W \cap h_{k,r}^{-1}(j)| = |W| = r$. Choose $s = r$ in Lemma 1.1., there exists a $k \in U$ such that

$$\sum_{j=1}^{r} B(r,W,k,j)^2 < \frac{2r^2}{r} + \sum_{j=1}^{r} B(r,W,k,j) = 2r + r = 3r.$$

This proves the first statement.

Now for the second, by choosing $s = r$ in Lemma 1.2, we have $\sum_{j=1}^{r} \binom{B(r,W,k,j)}{2} < 2r$ and using the fact $\sum_{j=1}^{r} B(r,W,k,j) = r$, we get $\sum_{j=1}^{r} B(r,W,k,j)^2 < 4r + \sum_{j=1}^{r} B(r,W,k,j) < 5r$. □

Corollary 3. *1. There exists a $k' \in U$ such that the function h_{k',r^2} is one-to-one when restricted to W.*

2. For at least half of the values $k' \in U$, the function $h_{k',2r^2}$ is one-to-one when restricted to W.

Proof. Setting $s = r^2$ in Lemma 1.1., there exists a $k' \in U$ such that $\sum_{j=1}^{r^2} \binom{B(r^2,W,k',j)}{2} < 1$, that is, for all j, $\binom{B(r^2,W,k',j)}{2} = 0$ which implies that $B(r^2,W,k',j) \leq 1$. In words, for all j, the number of times the value j is attained by the function h_{k',r^2} when restricted to W is at most 1, that is, the function h_{k',r^2} is one-to-one when restricted to W.

Setting $s = 2r^2$ in Lemma 1.2., we get $\sum_{j=1}^{2r^2} \binom{B(s,W,k',j)}{2} < 1$. As seen in the first part, this implies that the function $h_{k',2r^2}$ is one-to-one when restricted to W. □

Such one-to-one functions will be called perfect hash functions.

Description of data structure

Suppose we want to represent a set $S \subseteq U$, $|S| = n$, $|U| = m$ in memory T. We assume that each cell of T can hold $\log m$ many bits. Next, we do the following.

STEP 1: *Partitioning the given set, and storing pointers in the first level.*

- Substitute $W = S$ and $r = n$, and find an appropriate $k \in U$ satisfying Corollary 2.1.. Store it in $T[0]$.
- Use this k (content of $T[0]$), to partition S into blocks W_j for each $j = 1, 2, \ldots, n$, where
  - $W_j = \{x \in S \mid h_{k,n}(x) = j\}$,
  - $\sum_{j=1}^{n} |W_j|^2 < 3n$ *by Corollary 2.1. since* $|W_j| = B(n, S, k, j)$.
- Let $T' = T[1], T[2], \ldots, T[n]$ be the cells assigned to each W_j (called primary cells)
- For each W_j, $j = 1, 2, \ldots, n$, assign a memory T_j of size $|W_j|^2 + 2$ to resolve W_j, and store the pointer to T_j in the cell $T[j]$.
  - *The total memory used so far is at most* $6n+1$ *(*1 *for* k*,* n *for pointers in primary cells, and* $5n$ *for secondary memory blocks).*

STEP 2: *Resolving each block using perfect hash function in the second level.*

Consider a block T_j where W_j is to be resolved.

- Store $|W_j|$ in the first location of T_j.
- Substitute $W = W_j$ and $r = |W_j|$, and find an appropriate $k'_j \in U$ satisfying Corollary 3.1.. Store it in the second location of T_j.
- Store each $x \in W_j$ in $\left(h_{k'_j, |W_j|^2}(x) + 2\right)^{th}$ location of T_j.
  - *Recall* $h_{k'_j, |W_j|^2}(x) = (k'_j x \bmod p) \bmod |W_j|^2$*, and note that +2 is due to the first two cells being already occupied.*

Query execution

Input: q.

- Set $k = T[0]$.
- Set $j = h_{k,n}(q)$.
- Access $T[j]$ (contains pointer to block T_j), and access the quantities in the first two locations of T_j which are $|W_j|$ and k'_j respectively.
- Set $l = h_{k'_j, |W_j|^2}(x) + 2$.

- Access l^{th} cell of T_j. $q \in S$ iff q is in this cell.

Note that processing a query requires accessing only 5 cells.

Construction time

The running time is dominated by finding k and k'_j for each $j = 1, \dots, n$. By Corollary 2.1., finding a k requires going over all elements in U such that it satisfies $\sum_{j=1}^{n} B(n,S,k,j)^2 < 3n$. The sum can be computed as follows:

- Maintain an array A (initially all entries zero) of length n indexed by $j = 1, 2, \dots, n$.
- For each $x \in S$, compute $h_{k,n}(x) = kx \bmod p \bmod n$, and increment the $h_{k,n}(x)^{th}$ entry of A.
- Now $A[j] = B(n,S,k,j)$ for each $j = 1, 2, \dots, n$.
- Compute the sum $\sum_{j=1}^{n} A[j]^2$, check whether it is less than $3n$.
- If yes, then pick k. Otherwise, move to another choice of $k \in U$.

For each k, this procedure takes $\mathcal{O}(n)$ time, and hence the worst case time to find a hash function given by k satisfying Corollary 2.1. is $\mathcal{O}(mn)$. Similarly, finding a perfect hash function given by k'_j for each W_j satisfying Corollary 3.1. takes time $\mathcal{O}(m|W_j|^2)$. Summing over all $j = 1, 2, \dots, n$, the total time to find all secondary perfect hash functions given by k'_j's is $\mathcal{O}(mn)$. So overall time is $\mathcal{O}(mn)$.

Improved construction time

The construction time can also be improved to $\mathcal{O}(n^3 \log m)$. For this, we need a lemma.

Lemma 4. *There exists a prime $q < n^2 \log m$ that does not divide any of the elements in S, and that separates these elements into distinct residue classes mod q.*

If $m < n^2 \log n$, then clearly $\mathcal{O}(nm) = \mathcal{O}(n^3 \log n) = \mathcal{O}(n^3 \log m)$. If $m \geq n^2 \log n$, then

- We produce a prime q satisfying Lemma 4 ($q < n^2 \log m$). It can be done in time $\mathcal{O}(nq)$.
- Store q at the location $T[-1]$.
- Use this prime q to map the elements $x_1, x_2, \dots, x_n$ of S to $x_1 \bmod q, x_2 \bmod q, \dots, x_n \bmod q$ respectively. By Lemma 4, all these elements are distinct.
- Now store each $x \in S$ as before but now replacing the value x with $x \bmod q$ in earlier computations.
- Then, the construction time for this new problem becomes $\mathcal{O}(nq) = \mathcal{O}(n^3 \log m)$.

Proof of Lemma 4. For $S = \{x_1, \dots, x_n\}$, we want an existence of a prime $q < n^2 \log m$ that does not divide any of x_i's, and no two elements in S have the same remainder when divided by q, i,e, if $t = \prod_i x_i \prod_{i<j}(x_i - x_j)$, then q must not divide t.

Since $x_i \le m$, we have $|t| \le m^n m^{\binom{n}{2}}$, i.e., $\log|t| \le \binom{n+1}{2} \log m$.

Suppose on the contrary that all primes $q < n^2 \log m$ divide t. Then, it must be that $\prod_{q<n^2 \log m, q \text{ prime}} q \le t$ which implies $\log(\prod_{q<n^2 \log m} q) \le \log|t| \le \binom{n+1}{2} \log m$.

Recall that if $f(x)$ is the number of primes less than x, then by prime number theorem, $f(x) = \frac{x}{\log x} + o\left(\frac{x}{\log x}\right)$, and it is also equivalent to, in terms of Chebyshev function, $\log(\prod_{q<x, q \text{ prime}} q) = x + o(x)$. Substitute x with $n^2 \log m$, we get $\log(\prod_{q<n^2 \log m, q \text{ prime}} q) = n^2 \log m + o(n^2 \log m)$.

But now we have $n^2 \log m + o(n^2 \log m) \le \binom{n+1}{2} \log m$, which can also be written as, $1 + o(1) \le \frac{1}{2}\left(1 + \frac{1}{n}\right)$. This a contradiction. So the Lemma is proved. □

To summarize, we constructed a data structure in time $\mathcal{O}(n^3 \log m)$ using space $6n+1$ and query time 5.

Expected construction time

Next, we can show that by allowing the space for T slightly more than before, we can show that the construction time is $\mathcal{O}(n)$ in expectation.

Now in storing S in memory T, we follow the same procedure as before except that we allocate $2|W_j|^2+2$ space for a block W_j and use primary hash function given by Corollary 2.2. in the first step and secondary perfect hash function $h_{k',2|W_j|^2}$ given by Corollary 3.2. in the second step. Hence, the overall space used for T is $13n+1$ ($n+1$ for primary cells, and at most $12n$ for secondary cells). Since the expected number of choices for k or k' until a suitable one is found is 2, the expected construction time becomes $\mathcal{O}(n)$.[2] We'll stick to this randomized construction idea in the next section as well.

3 An even more refined construction

First observe that there are three cases for a block W_j:

Case 1: $|W_j| = 0$.

- Then, we are allocating 3 cells for such a block.
- Perhaps, we can just use a single bit instead to check whether a block is empty or non-empty?

Case 2: $|W_j| = 1$.

- Then, we are allocating 4 cells for such a block.
- Perhaps, we can simply store the element of W_j in the primary cell itself?
- For this, we also need a tag bit to distinguish between $|W_j| = 1$ and $|W_j| \ge 2$ if W_j is nonempty.

[2] Note that the worst case time can be infinite.

- Note that in the worst case if all W_j's are singleton sets, then we already have used at least $n+1$ cells for elements and $n/\log m = o(n)$ cells for tag bits.

Case 3: $|W_j| \geq 2$.

- Then, we'll resolve only such a block in secondary memory of size $2|W_j|^2 + 2$ as before.
- Perhaps, we can show that the total memory used to resolve such blocks is $o(n)$?
- For this, we'll partition S into a large number of blocks, say $g(n)$ instead of n as done earlier. Since $g(n)$ will be larger compared to n, hopefully there will be very few blocks with more than one element such that the total space required is $o(n)$.

Lemma 5. *There exists a $g(n)$ such that the space to resolve blocks with more than one elements is $o(n)$.*

Proof. Recall that by Lemma 1.2., for half of the values of $k \in U$, we have $\sum_{j=1}^{s} \binom{B(s,W,k,j)}{2} < \frac{2r^2}{s}$. Choose $W = S$, $s = g(n)$ and $r = n$, then

$$\sum_{j=1}^{g(n)} \binom{B(g(n), S, k, j)}{2} < \frac{2n^2}{g(n)},$$

and since W_j will be such that $|W_j| = B(g(n), S, k, j)$,

$$\sum_{j=1}^{g(n)} \binom{|W_j|}{2} < \frac{2n^2}{g(n)}.$$

Note that all those terms for which $|W_j| \leq 1$ contribute zero to the sum. We need a fact $x^2 \leq 4\binom{x}{2}$ for $x \geq 2$ (Proof: $x^2 \leq c\binom{x}{2}$ holds for $c \geq 2 + \frac{2}{x-1}$. Choose $c = 4$.) using which we have a $k \in U$ such that

$$\sum_{\substack{j=1,\\ |W_j|\geq 2}}^{g(n)} |W_j|^2 \leq \sum_{\substack{j=1,\\ |W_j|\geq 2}}^{g(n)} 4\binom{|W_j|}{2} < 4\frac{2n^2}{g(n)}.$$

This implies

$$\sum_{\substack{j=1,\\ |W_j|\geq 2}}^{g(n)} |W_j|^2 < 8\frac{n^2}{g(n)},$$

and by choosing $g(n)$ such that $\lim_{n\to\infty} \frac{n}{g(n)} = 0$, the sum is $o(n)$. Next, since $|W_j| \geq 2$,

$$\sum_{\substack{j=1,\\ |W_j|\geq 2}}^{g(n)} 2 \leq \sum_{\substack{j=1,\\ |W_j|\geq 2}}^{g(n)} |W_j|^2 = o(n),$$

where the sum on the left is exactly the number of $\#W_j$'s of size at least 2. Thus,

$$\sum_{\substack{j=1,\\|W_j|\geq 2}}^{g(n)} (2|W_j|^2+2) = 2\sum_{\substack{j=1,\\|W_j|\geq 2}}^{g(n)} |W_j|^2 + \sum_{\substack{j=1,\\|W_j|\geq 2}}^{g(n)} 2 = o(n)+o(n) = o(n).$$

□

We choose $g(n) = n\sqrt{\log n}$ for convenience.

Description of data structure

Suppose we want to represent a set $S \subseteq U$, $|S| = n$, $|U| = m$ in memory T. We do the following.

STEP 1: *Partitioning the given set, and storing pointers to nonempty blocks in the first level along with tag bits to distinguish between two types of nonempty blocks.*

- Set $g(n) = n\sqrt{\log n}$.
- Substitute $W = S$, $s = g(n)$ and $r = n$. Pick $k \in U$ u.a.r, and check if it satisfies Lemma 1.2. Repeat until a suitable one is found. In that case, store it in $T[0]$.
- Use this k (content of $T[0]$), to partition S into blocks W_j, $j = 1,2,\ldots,g(n)$, where
  - $W_j = \{x \in S \mid h_{k,g(n)}(x) = j\}$.
  - Note $\sum_{\substack{j=1,\\|W_j|\geq 2}}^{g(n)} |W_j|^2 = o(n)$ by Lemma 5, since $|W_j| = B(g(n), S, k, j)$.
- Let $T' = T[1], T[2], \ldots, T[n']$ be the cells assigned to each nonempty W_j's in increasing order of j, where n' is the number of nonempty blocks (primary cells).
  - *Note that $T[n']$ may even be associated with the block $W_{g(n)}$ if nonempty.*
- Let C be a sequence of n tag bits used to distinguish between the cases $|W_j| = 1$ and $|W_j| \geq 2$.
  - *These tag bits can be packed in $n/\log m \leq n/\log n = o(n)$ cells.*
- For each nonempty W_j, $j = 1,\ldots,g(n)$:
  - If $|W_j| = 1$, then set $C[j] = 0$, and store the single item of W_j in the next available cell of T' directly.
  - If $|W_j| \geq 2$, then set $C[j] = 1$, and assign a memory T_j of size $2|W_j|^2+2$ to resolve W_j, and store the pointer to T_j in the next available cell in T'.

STEP 2: *Resolving each block of size at least 2 using a perfect hash function in the second level.*

Consider memory T_j to resolve W_j

- Store $|W_j|$ in the first location of T_j.

- Substitute $W = W_j$ and $r = |W_j|$. Pick $k'_j \in U$ u.a.r, and check if it satisfies Corollary 3.2. Repeat until a suitable one is found. In that case, store it in the second location of T_j.
- Store each element $x \in W_j$ in $\left(h_{k'_j, 2|W_j|^2}(x) + 2\right)^{th}$ location of T_j.
  - *As seen earlier, the total memory allocated for all W_j's with at least two elements is $o(n)$.*

STEP 3: *Setting up an auxillary data structure to check whether a block is nonempty and find tag bits and primary cells associated with it.*

- Let $t = (g(n)/n)^2$, and partition $I = [1, g(n)]$ into $g(n)/t$ intervals $\sigma_1, \ldots, \sigma_{g(n)/t}$ each of size t.
- Let B be an array of size $g(n)/t$, a cell for each interval.
- For each interval σ, set $B[\sigma]$ to be the address of the location immediately preceding the cells in T' associated with the first nonempty W_j for $j \in \sigma$.
  - *Note $B[\sigma] + 1$ gives the index where the interval containing the primary cell of W_j begins.*
  - *Note that the size of B is $g(n)/t = n^2/g(n) = n/\sqrt{\log n} = o(n)$.*
- Let A be a sequence of bits of total length $g(n)\log t$ to store offsets, where each portion is of size $\log t$ and corresponds to some $j \in [1, g(n)]$.
- For each $j = 1, 2, \ldots, g(n)$:
  - If W_j is empty, set $A[j] = 0$.
  - Else find the interval σ such that $j \in \sigma$, and set $A[j]$ to be the index of j in the interval σ (offset).
- *Thus, $B[\sigma] + A[j]$ gives the exact location of the primary cell of block W_j.*
  - *Note any $A[j]$ has size $\log t$ since it is an index in an interval of length t*
  - *Since A has total length $g(n)\log t$, and any cell of the memory can hold at most $\log m$ bits, the number of cells required to store A is $\mathcal{O}(g(n)\log t/\log m) = \mathcal{O}(n \log\log n/\sqrt{\log n}) = o(n)$.*

Query execution

Input: q

- Set $k = T[0]$.
- Set $j = h_{k,g(n)}(q)$.
- Access $A[j]$, the j^{th} portion of size $\log t$ of A.
  - If $A[j] = 0$, then return NO.

- Else, access $B[\sigma]$ and compute $j' = B[\sigma] + A[j]$ where σ is the quotient of j when divided by t (interval size).

- Access tag bit $C[j']$.
  - If $C[j'] = 0$, then access $T[j']$ (contains pointer to singleton block W_j). Return YES iff q is in this cell.
  - Else, access $T[j']$ (contains pointer to memory T_j where W_j is resolved), and access the quantities in the first two locations of T_j which are $|W_j|$ and k'_j respectively of size $2|W_j|^2 + 2$

- Set $l = h_{k'_j, 2|W_j|^2}(q) + 2$.

- Access l^{th} cell of T_j. Return YES iff q is in this cell.

Note that processing a query requires accessing at most 7 cells. The space used for primary cells is at most $n+1$, for secondary cells is $o(n)$ and for each of A, B and C is also $o(n)$. So the overall space is $n + o(n)$. The construction time is $\mathcal{O}(n)$ in expectation as seen in the previous section.

References

[1] M. Fredman, J. Komlós, E. Szemerédi, *Storing a Sparse Table with* $O(1)$ *Worst Case Access Time*, Journal of the ACM, 31(3):538-544, 1984.