Hashing III : Perfect Hashing[1]

- We saw that universal hash functions allow us to solve the static dictionary problem of searching in a set $D \subseteq U$ of m elements from a universe of N elements in $O(1)$ query time, and $O(m)$ space. However, the query time is in expectation. That is, for any $x \in U$, the expected time (given h) to search is $O(1)$. In particular, if an "adversary" knew the function h, then they could find an x for which $\textsc{Search}(x)$ took more than constant time. In this lecture we see an idea of ***double hashing*** which allows one to obtain an *worst-case* $O(1)$ time result.

- *A large space solution.* Before describing the double-hashing idea, let us show an $O(1)$ worst-case solution which takes $O(m^2)$ space. The main ideas are from the *birthday paradox* problem: if one throws $\leq \sqrt{n}/2$ balls into n bins, then constant probability there is no collision.

  Let H be a UHF of functions $h : U \to [m^2]$, where we use $[k]$ as a shorthand for $\{0, 1, \ldots, k-1\}$. For $x, y \in D$ with $x \neq y$, let $Z_{x,y}$ be the indicator random variable that $h(x) = h(y)$ when $h \in_R H$ is drawn uar. Let $Z := \sum_{(x,y)\in D\times D, x\neq y} Z_{x,y}$ denote the number of collisions. By the property of UHF, we know that $\mathbf{Pr}[Z_{x,y} = 1] \leq \frac{1}{m^2}$. Thus,

$$\mathbf{Exp}[Z] \leq \binom{m}{2} \cdot \frac{1}{m^2} < \frac{1}{2} \quad \Rightarrow \quad \mathbf{Pr}[Z = 0] \geq \frac{1}{2}$$

  In plain English, if we draw an $h \in H$ uar, then the probability we get a *perfect* hash function with no collisions is $\geq \frac{1}{2}$. Thus, the pre-processing step of "keep sampling $h \in H$ till we get a perfect hash function" takes $O(m)$ time. And once we get a perfect hash function, then $\textsc{Search}(x)$ is $O(1)$ time worst-case.

- *Double Hashing.* Recall the hashing solution we had. We hashed $x \in D$ to $T[h(x)]$ where $T[h(x)]$ was a list. In expectation, this list size was small, but some lists could indeed be big, and therefore, in worst-case, the search time is not $O(1)$. The idea of double hashing is simple: instead of using a list to store $T[h(x)]$, use *another* hash-function. Except this second hash-function is going to be a *perfect* hash function for the smaller dictionary of the items that get mapped to $T[h(x)]$. If b_i elements get mapped to $T[h(x)]$, then from the previous bullet point, the space required would be $O(m)$ (for the first hash function) and $\sum_{i=1}^{n} O(b_i^2)$ for the n secondary hash-functions. Since we don't expect any b_i to be very large, the sum of squares can be bounded by $O(m)$. This is the high level idea, and now we give details.

- *Construction using UHFs.* We are going to draw our first-level hash function (the primary hash function) from a UHF family mapping U to $n := m$.

  For $1 \leq i \leq n$, define b_i to be the number of $x \in D$ with $h(x) = i$. As discussed above, we wish to argue that $B := \sum_{i=1}^{n} b_i^2$ is small. Indeed, we can show $\mathbf{Exp}[B]$ is small as follows.

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 11th April, 2021*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

Define $C_i := \binom{b_i}{2}$ denote the number of *pairs* of distinct x and y which map to the position i. Define $C := \sum_{i=1}^{n} C_i$ to be the total number of collisions. Now note that

$$C = \sum_{(x,y) \in D \times D : x \neq y} Z_{x,y}$$

where $Z_{x,y}$ is the indicator random variable of the event $h(x) = h(y)$. Since h is drawn from a UHF, we get that

$$\mathbf{Exp}[C] \le \binom{m}{2} \cdot \frac{1}{n} = \frac{m-1}{2} \quad \text{since } n = m$$

Now, we get

$$\mathbf{Exp}[B] = \mathbf{Exp}\left[\sum_{i=1}^{n} b_i^2\right] = \mathbf{Exp}\left[\sum_{i=1}^{n} \left(b_i + 2 \cdot \binom{b_i}{2}\right)\right] = \underbrace{\mathbf{Exp}[\sum_{i=1}^{n} b_i]}_{=m} + 2 \cdot \underbrace{\mathbf{Exp}[C]}_{\le \frac{m-1}{2}} < 3m$$

And thus, by Markov's inequality

$$\mathbf{Pr}[B \ge 6m] \le \frac{1}{2}$$

Therefore, in $O(1)$ samples of h from the UHF family, we can obtain one with $B \le 6m$. And then, we simply apply the perfect hash functions from the second bullet point.

- *Algorithm Details.* Now we are ready to describe the pre-processing and the SEARCH algorithm.

procedure PREPROCESS(D): ▷ *$|D| = m$.*
  while true **do**:
    Draw h from a strongly UHF which takes U to $[n]$ where $n = m$.
    Evaluate b_i which is the number of $x \in D$ mapping to i, for all $i \in [n]$. ▷ *$O(m)$ time.*
    if $\sum_{i=1}^{n} b_i^2 > 6m$ **then**: ▷ *This occurs with probability $\le \frac{1}{2}$.*
      Abort this loop and go to next loop.
  ▷ *At this point, we know $\sum_{i=1}^{n} b_i^2 \le 6m$.*
  for $1 \le i \le n$ **do**:
    Let $D_i := \{x \in D : h(x) = i\}$ with $b_i = |D_i|$.
    Construct *perfect* hash function $g_i : U \to [b_i^2]$ as in second bullet point for D_i.
    Construct the corresponding hash table $T_i[0 : b_i^2 - 1]$
    ▷ *This takes $O(b_i)$ time in expectation, and uses $O(b_i^2)$ space.*
    Store x in location $T_i[g_i(x)]$.
  ▷ *The total time and space taken over the for-loops is $O(m)$ time.*

To search for a given $x \in U$, we first compute $i := h(x)$, and then search for x in $T_i[g_i(x)]$. This takes $O(1)$ time if function computations and accesses are $O(1)$ time.

- *Space Analysis.* The total space required in the hash-tables are $\sum_{i=1}^{n} b_i^2 \le 6m$ by design. The total time taken to find the g_i's is $\sum_{i=1}^{n} O(b_i) = O(m)$.