Preliminary

Dictionary
Let $U$ be the universe set of strings, $|U| = N$ (very large). We want to build a dictionary data structure to map $W \subseteq U$ ($|W| = n \ll N$) to its relevant information. The dictionary supports a lookup operation which is given $x \in U$ determine if $x \in W$ (look up associated data).

Types of dictionaries

- A static dictionary only supports lookup once it is built.
- A dynamic dictionary should also supports insertion and deletion.

Hash Table
A hash table is a length $m$ array with an associated hash function $h$ so that $x$ is stored in the $h(x)$-th element. Hash function $h$ maps $x \in U$ to a arbitrary location of the hash table. Collision of hash function $h$ happens when $h(x) = h(y), x \neq y, x, y \in W$.

Need a function that outputs “kind of randomly” to prevent adversarial examples causing serious collisions but the function has to have a the same output every time it is fed with the same input.

Hash family
Hash family is a collection of hash functions $h : U \rightarrow [m]$

k-wise independence hash family
Let $x_i \in W, y_i \in [m]$ and $x_i$ are all distinct.

Fix $x, y$, and randomly choose $h$ from the k-wise independence family.
\[ Pr_h(h(x_1) = y_1 \land \ldots \land h(x_k) = y_k) = \frac{1}{m^k} \]

Note that k-wise independence imply (k-1)-wise independence.

2-wise independence hash family
\[ Pr_h(\#(x \neq y) : x, y \in W, h(x) = h(y)) = O\left(\frac{n^2}{m}\right) \]

### Dealing with collisions

Easy answer: Array of length \( m \) buckets, each bucket in the array stores a doubly linked list of \( x \in W \) s.t. \( h(x) \) the same value. Assume \( h \) is from 2-wise independence hash family.

The time to build this static dictionary requires \( O(n) \) time.

From the 2-wise independence assumption: \( Pr_h(h(x) = h(x')) = \frac{1}{m} \). The lookup time for a given \( x = O(1) + O(E[\# \ of \ x' \in W \ s.t. \ h(x) = h(x')] \Rightarrow E[time \ per \ lookup] = O(1 + \frac{n}{m}) \).

If \( m \gg n \). it has a constant expected time per lookup.

**Dynamic**

Analysis for insert/delete is the same as lookup. First perform a lookup find the linked list and insert/delete the element in \( O(1) \).

**Problems**

- In the worst case, what if \( h \) from the 2-independent family has all collisions send to the all choices have a bucket with \( \sqrt{n} \) element in it.
- Also possible with \( \frac{1}{n} \) probability (over the choice of \( h \)) that all elements collide in the same bucket.

**Perfect Hashing** (Static dictionary)

We want all lookups done in \( O(1) \) time, not the average case.

**Easy case** With a large enough \( m \). If \( m \geq 2\binom{n}{2} \approx n^2 \)

With \( E[\# \ pairs \ that \ collides] = \binom{n}{2}/m \leq \frac{1}{2} \), we can try \( \approx 2 \) hash functions and the chance of the pair collide in both hash function is small.

**Idea** find \( h \) that is twice as long, \( h : W \to [2n] \) without collisions

Even if \( h \) is uniform random (weaker assumption than k-wise independence), the probability of collision \( = 1(1 - \frac{1}{2n})(1 - \frac{3}{2n}) \ldots (1 - \frac{n-1}{2n}) \leq exp\left(\frac{n}{10}\right) \) is exponentially small
2-stage hash function

Generate the 1-st hash function \( h_o : U \to [m] \). In bucket i, stores the second stage hash function \( h_i : U \to [(\# \text{ elements in bucket } i)^2] \). If there is a few collisions in first hash function, we can choose a second-stage hash function on a smaller set of elements. To determine final hash function \( h(x) \) look in bucket \( h_{h_o(x)}(x) \) of the \( h_o(x) \)-th bucket.

Implementation

Pick \( h_o \)
hash all elements \( x \in W \)
compute \( c_i = ( \# \text{ elements } x \in W : h_o(x) = i) \)
if \( \sum c_i^2 \gg n \) try again
else
allocate \( c_i^2 \) memory in the \( i \)-th array
for \( i = 1 \) to \( m \)
try \( h_i \)
hash all \( c_i \) elements
if collision try a new \( h_i \)

- \( \mathbb{E}[\text{build time}] = O(n) \)
- Lookup time worst case is \( O(1) \) which is the run time of two hash functions.
- memory usage \( O(m) + \sum c_i^2 = O(m) + O(n) + O\left(\frac{n^2}{m}\right) = O(n) \)

\[
Pr(h_o(x_1) = h_o(x_2) = i) = \begin{cases} 
\frac{1}{m^2}, & \text{if } x_1 \neq x_2 \\
\frac{1}{m}, & \text{if } x_1 = x_2 
\end{cases}
\]

\[
\mathbb{E}[\sum c_i^2] = \sum_i \sum_{x_1, x_2} Pr(h_o(x_1) = h_o(x_2) = i) = O(n) + O\left(\frac{n^2}{m}\right)
\]

\((O(n) \text{ from the cases of } x_1 = x_2, O\left(\frac{n^2}{m}\right) \text{ from the cases of } x_1 \neq x_2)\)