Dictionaries and Hashing recap

- A dynamic dictionary stores (key, value)-pairs and supports:
  - add(key, value), lookup(key) (which returns value) and delete(key)
- Hash table \( T \) of size \( m \geq n \).
- Hash function maps a key \( x \) to position \( h(x) \), i.e. \( T[h(x)] = (key, value) \).
- Collisions are fixed by chaining

A set \( H \) of hash functions is weakly universal if for any two keys \( x, y \in U \) (with \( x \neq y \)),
\[
\Pr \left( h(x) = h(y) \right) \leq \frac{1}{m}
\]
(h is picked uniformly at random from \( H \)).

Using weakly universal hashing:
For any \( n \) operations, the expected run-time is \( O(1) \) per operation.
But this doesn’t tell us much about the worst-case behaviour.

Perfect hashing - a first attempt

A set \( H \) of hash functions is weakly universal if for any two keys \( x, y \in U \) (\( x \neq y \)),
\[
\Pr \left( h(x) = h(y) \right) \leq \frac{1}{m}
\]
where \( h \) is picked uniformly at random from \( H \).

Step 1: Insert everything into a hash table of size \( n \)
using a weakly universal hash function

Step 2: Check for collisions

Step 3: Repeat if necessary

How many collisions do we get on average?
\[
\mathbb{E}(C) = \mathbb{E}\left(\sum_{x \neq y, x < y} I_{x,y}\right) = \sum_{x \neq y, x < y} \mathbb{E}(I_{x,y}) \leq \sum_{x \neq y, x < y} \frac{1}{m} = \left(\frac{n^2}{2}\right) \frac{1}{m} \leq \frac{n^2}{2m} \leq \frac{n}{2}
\]
where indicator random variable \( I_{x,y} = 1 \) iff \( h(x) = h(y) \).

Expected construction time

Step 1: Insert everything into a hash table of size \( m = \frac{n^2}{2} \)
using a weakly universal hash function

Step 2: Check for collisions

Step 3: Repeat if there was a collision

How many times do we repeat on average?
\[
\mathbb{E}(C) \leq \frac{1}{2}
\]
Markov’s inequality
The expected number of collisions: \( \mathbb{E}(C) \leq \frac{1}{2} \)
The probability of at least one collision: \( \Pr(C \geq 1) \leq \frac{1}{2} \)

The probability of zero collisions is at least \( \frac{1}{2} \)
i.e. at least as good as tossing a heads on a fair coin
\[
\mathbb{E}(\text{runs}) = \mathbb{E}(\text{coin tosses to get a heads}) = 2
\]
\[
\mathbb{E}(\text{construction time}) = O(m) \cdot \mathbb{E}(\text{runs}) = O(m) = O(n^2)
\]

... and then the look-up time is always \( O(1) \)
(because any \( h(x) \) can be computed in \( O(1) \) time)
How much space does this use?

The size of $T$ is $O(n)$

The size of $T_i$ is $O(n_i^2)$

Storing $h_i$ uses $(O(1))$ space

So the total space is...

$O(n) + \sum_i O(n_i^2) = O(n) + O\left(\sum_i n_i^2\right)$

Perfect Hashing - Summary

Step 1: Insert everything into a hash table, $T$, of size $n$ using a weakly universal (w.u.) hash function, $h$

Step 2: The $n_i$ items in $T[i]$ are inserted into another hash table $T_i$ of size $n_i^2$ using w.u hash function $h_i$

(Step 3) Immediately repeat if either

a) $T$ has more than $n$ collisions
b) some $T_i$ has a collision

Two questions remain:

What is the expected construction time?

What is the space usage?

Perfect Hashing - Space usage

Step 1: Insert everything into a hash table, $T$, of size $n$ using a weakly universal (w.u.) hash function, $h$

Step 2: The $n_i$ items in $T[i]$ are inserted into another hash table $T_i$ of size $n_i^2$ using w.u hash function $h_i$

(Step 3) Immediately repeat if either

a) $T$ has more than $n$ collisions
b) some $T_i$ has a collision

How much space does this use?

The size of $T$ is $O(n)$

The size of $T_i$ is $O(n_i^2)$

Storing $h_i$ uses $(O(1))$ space

So the total space is...

$O(n) + \sum_i O(n_i^2) = O(n) + O\left(\sum_i n_i^2\right)$

Perfect Hashing - Attempt three

Step 1: Insert everything into a hash table, $T$, of size $n$

Step 2: Using a weakly universal (w.u.) hash function, $h$

... but do not use chaining

Perfect Hashing - Expected Construction time

Step 1: Insert everything into a hash table, $T$, of size $n$

Step 2: Using a weakly universal (w.u.) hash function, $h$

... but do not use chaining

How do we compute?

The expected number of collisions: $E(C) \leq \frac{n}{2}$

The probability of at least $n$ collisions: $Pr(C \geq n) \leq \frac{1}{2}$

The probability of at most $n$ collisions is at least $\frac{1}{2}$

i.e. at least as good as tossing a heads on a fair coin

$E(\text{runs}) \leq E(\text{coin tosses to get a heads}) = 2$

$E(\text{construction time}) = O(n) + E(\text{runs}) = O(n)$

... but the look-up time could be rubbish (lots of collisions)

Two questions remain:

What is the expected construction time?

What is the space usage?

Perfect Hashing - Expected construction time

Step 1: Insert everything into a hash table, $T$, of size $n$

Step 2: Using a weakly universal (w.u.) hash function, $h$

... but do not use chaining

How much space does this use?

The size of $T$ is $O(n)$

The size of $T_i$ is $O(n_i^2)$

Storing $h_i$ uses $(O(1))$ space

So the total space is...

$O(n) + \sum_i O(n_i^2) = O(n) + O\left(\sum_i n_i^2\right)$

The overall expected construction time is therefore:

$E(\text{construction time}) = E\left(\text{construction time of } T + \sum_i E(\text{construction time of } T_i)\right)$

$= E(\text{construction time of } T) + \sum_i E(\text{construction time of } T_i)$

$= O(n) + \sum_i O(n_i^2) = O(n) + O\left(\sum_i n_i^2\right) = O(n)$

The FKS hashing scheme:

- Has no collisions
- Every lookup takes $O(1)$ worst-case time.
- Uses $O(n)$ space.
- Can be built in $O(n)$ expected time.

The look-up time is always $O(1)$

1. Compute $i = h(x)$ ($x$ is the key)
2. Compute $j = h_i(x)$
3. The item is in $T[j]$