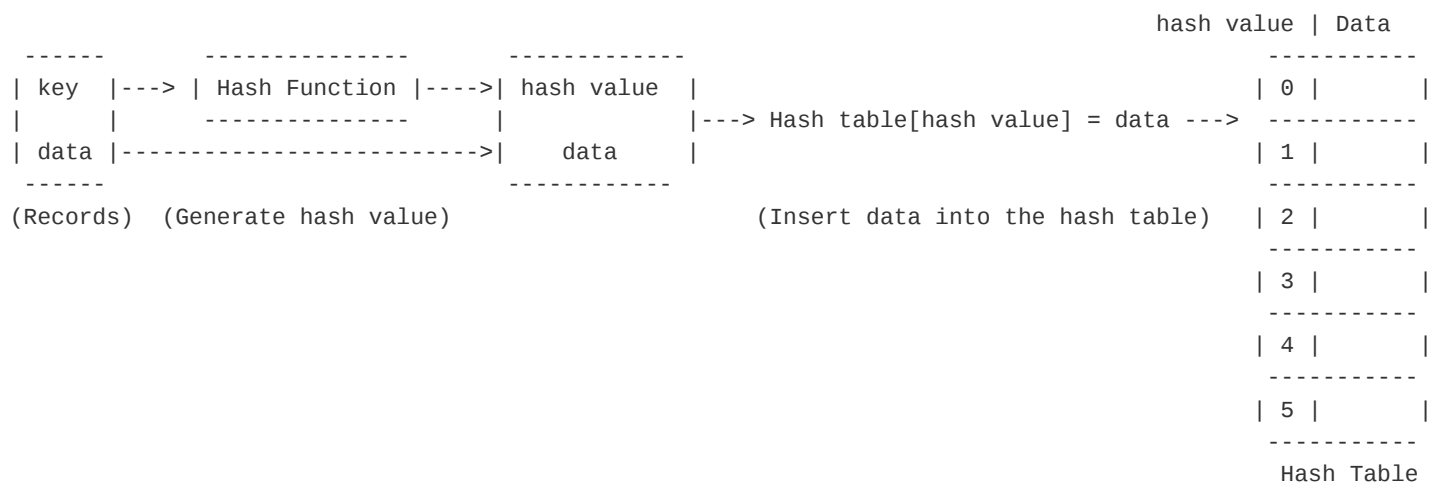
Hash Table

Hash table stores n data in an array of a fixed length m. Similar to a list which inserts/finds elements using integer indices i, Hash table inserts/finds elements using integer hash value. The process of insertion is called $hashing$. Hash value is generated from a hash function that takes in key associated with the data and return an integer value. Thus, a key $(str\ or\ int)$ indexes a data item into an array. Here is a schematic:



Hash tables are designed for $O(1)$ $insert$ and $find$ operations.

Hash Functions

Hash function generates hash value that indexes items into an array. This begs an important question:

- What happens when two or more different keys return same hash value? (Collision, we call it)

It is impossible to generate a perfect hash function because in most cases the distribution of input data in unknown. Collisions are inevitable and can only be mitigated. A first step to it is to use a good hash function with the following properties:

- Computable in $O(1)$
- Generates hash values that are uniformly distributed across the hash table
- Utilizes all the slots in the table

There are two strategies for collision resolution:

- **Separate Chaining:** Use a chain (e.g linked list) to store multiple data items
- **Open Addressing or Probing:** Search for empty slot using a second function.

Separate Chaining

A chain (e.g linked list) is maintained in each slot that stores multiple items hashed to the same value. Load factor α is defined as the expected number of elements stored in a chain.

Assuming simple uniform hashing, load factor $\alpha = n/m$

Consider a hash table H of size m where n data items need to be stored. Assume a simple uniform hashing, i.e any key k not already hashed is equally likely to be hashed to any m slots. Thus, without loss of generality, we can pick a slot and find its expected length.

Let X_i be the indicator random variable that denotes whether ith item is hashed into the slot 1.

$$X_i = \begin{cases} 1 & if\ hashed\ to\ slot\ 1 \\ 0 & else \end{cases}$$

The expected value to X_i is given as,

$$E[X_i] = 1 \times P(X_i = 1) + 0 \times P(X_i = 0)$$

$$E[X_i] = P(X_i = 1)$$

The probability that the ith item is in slot $1 = 1/m$. Thus,

$$E[X_i] = 1/m$$

Let the random variable X be the length of slot 1. Then,

$$X = X_1 + X_2 + X_3 + ... + X_i + ... + X_n$$

Expected length of slot 1 is given as,

$$E[X] = E[X_1 + X_2 + X_3 + ... + X_n]$$

Using linearity of expectation,

$$E[X] = E[X_1] + E[X_2] + ... + E[X_n]$$

$$E[X] = 1/m + 1/m + ... + 1/m = n/m$$

Hence the load factor of hash table H is n/m

Assuming simple uniform hashing, when separate chaining is used for collision resolution, an unsuccessful search takes $\Theta(1 + \alpha)$ on average.

Let random variable X be the number of operations needed to search unsuccessfully for a key k and $E[X]$ its expected value. Since it is an unsuccessful search, entire chain will be searched. Thus,

X = Number of operations to generate hash value + Length of the chain
The expected number of operations is given as,

$$E[X] = O(1) + \alpha = \Theta(1 + \alpha)$$

Assuming simple uniform hashing, when separate chaining is used for collision resolution, a successful search takes $\Theta(1 + \alpha)$ on average.

Let random variable X be the number of operations needed to search successfully for a key k and $E[X]$ its expected value. Thus,

$X = Number\ of\ operations\ for\ hash\ value + Number\ of\ elements\ searched\ in\ the\ chain$
$X = O(1) + Number\ of\ elements\ searched\ in\ the\ chain$

$E[X] = 1 + E[Number\ of\ elements\ searched\ in\ the\ chain]$

Since elements are inserted at the head of the chain,
$E[Number\ of\ elements\ searched\ in\ the\ chain] = 1 + E[Average\ number\ of\ keys\ inserted\ after\ key\ k]$

It follows,
$E[X] = 2 + E[Average\ number\ of\ keys\ inserted\ after\ key\ k]$

Let k_i and k_j be the keys of ith and jth items hashed, where $i \in [1, n]$ and $j = i + 1$. Let X_{ij} be the indicator random variable such that:

$$X_{ij} = I(h(k_i) = h(k_j)), \quad h()\ is\ a\ hash\ function$$

Under simple uniform hashing,

$$E[X_{ij}] = 1 \times P(X_{ij} = 1) = P(h(k_i) = h(k_j)) = 1/m$$

Let S_i be the number of keys inserted after the key k_i

$$S_i = \sum_{j=i+1}^{n} X_{ij}$$

Average number of keys inserted after key k is

$$\frac{1}{n} \times \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$$

The expected value of the average number of keys inserted after key k is

$$E\left[\frac{1}{n} \times \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}\right]$$

Using linearity of expectation,

$$\frac{1}{n} \times \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}]$$

$$\frac{1}{n} \times \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{m}$$

$$\frac{1}{nm} \times \sum_{i=1}^{n} n - i$$

$$\frac{1}{nm} \times \left[\sum_{i=1}^{n} n - \sum_{i=1}^{n} i\right]$$

$$\frac{1}{nm} \times \left[n^2 - \frac{n(n+1)}{2}\right]$$

$$\left[\frac{\alpha}{2} - \frac{\alpha}{2n}\right]$$

Putting all together we get,

$$E[X] = 2 + \left[\frac{\alpha}{2} - \frac{\alpha}{2n}\right]$$

Thus, it takes $\Theta(1+\alpha)$ to successfully search for a key. What this tells us is that, if n is comparable to m, then search takes $O(1)$ time.

$$\alpha = n/m = O(m)/m = O(1)$$

Open Addressing / Probing

When collision occurs, scan rest of the table for an empty slot and insert item on the first empty slot found.
There are three methods of probing.

Probing	Hash Function
Linear Probing	$h(k) = (h'(k) + i) \ \% \ m$
Quadratic Probing	$h(k) = (h'(k) + i^2) \ \% \ m$
Double Hashing	$h(k) = (h_1(k) + ih_2(k)) \ \% \ m$

$h(k)$ is the hash function and $h'(k)$, $h_1(k)$ and $h_2(k)$ are auxiliary hash functions.

Linear Probing

- Linear probing suffers from primary clustering because you no longer have uniform hashing assumption valid. An empty slot that is preceded by i occupied slots has probability of getting filled $(i+1)/m$ and not $1/m$.
- For each slot, there is only one sequence of probe. Thus, there are only m unique probe sequences out of possible $m!$ sequences. Eg: for slot k, it only probes in sequence $k+1$, $k+2$, $k+3$ ...
- Linear probing is guaranteed to find an empty slot if there is one.

Quadratic Probing

- Quadratic probing suffers from secondary clustering for the fact that it also, like linear probing, has only m possible sequences. Keys that have same hash value are going to check for the same sequence of slots for an empty slot.
- For each slot, there is only one sequence of probe. Thus, there are only m unique probe sequence out of possible $m!$ sequences. Eg: for slot k, it only probes in sequence $k+1$, $k+4$, $k+9$, ...
- Quadratic probing does not always guarantee an empty slot because it might not search entire table. If table size is 16, alternate positions will be only 1, 4 or 9 distances away from the slot. However, we can guarantee an empty slot if hash table maintains two properties:
  - Hash table size m is prime
  - Hash table is at least half empty, load factor = $\alpha = n/m < 0.5$

  We will show that if $\alpha < 0.5$ and m is prime, then first $m/2$ probes are distinct and thus will find an empty slot. For sake of contradiction, let's say ith and jth probe among first $m/2$ probes point to the same slot.

$$h(x) + i^2 = h(x) + j^2 \quad \% \ m, \quad i \neq j$$

$$(i-j)(i+j) = 0 \quad \% \, m$$

Since $i \neq j$, $(i-j) \neq 0 \quad \% \, m$.
Since $i, j < m/2$ and m is prime, $(i+j) \neq 0 \quad \% \, m$
Hence, ith and jth probes must be distinct. It follows immediately that first $m/2$ probes are distinct, including $i = 0$.

Double Hashing

- Instead of a predefined i or i^2 steps, probe distances are calculated using auxiliary hash function $h_2(k)$. For a given key k, $h_2(k)$ can generate m' (prime less than m used in h_2 function) probe distances. Thus, double hashing provides mm' possible probe sequences out of possible $m!$. If m' is comparable to m, m^2 possible probe sequences are possible! And this is why it performs better than linear or quadratic probing.
- It should be noted that m' and m should be at least relatively prime. Imagine a situation where $h_2(k)$ returns m' and if m is divisible by m', then the number of distances to probe are severely compromised.
- It is obvious but should be noted that $h_2(k)$ should never return 0, as this means no probing at all.
- Double hashing does not suffer from clustering as is in the case for linear and quadratic probing. This is because each slot has multiple probe sequences possible. (m' to be exact)

Assuming simple uniform hashing, when open-addressing is used for collision resolution, an unsuccessful search takes at most $1/(1-\alpha)$ and $\alpha < 1$.

Consider searching for an item x whose hash value generated is j in a hash table containing n items in m slots, with $n < m$.

```
Consider this illustration that demonstrates linear probing.

probes:   2nd probe    3rd probe    4th probe                ith probe
         ----->-----  ----->-----  ----->-----  ----->-----  ----->-----
        /           \/           \/           \/           \/           \
slots:  j          j+1          j+2          j+3          ...          j+i-1
```

Few points to notice from the illustration:

- ith probe happens only after $i-1$ probes have occurred.
- $2nd$ probe happens when the first slot (jth in this case) is occupied. The probability that the slot is occupied = n/m, since there are n items that can be placed in m slots.
- $3rd$ probe happens when $2nd$ probe occurred leading to an occupied slot. The probability of $3rd$ probe = $(n/m)(n-1/m-1)$, since for $2nd$ slot there are $n-1$ items that can be placed in $m-1$ slots.

Thus, the probability of ith probe P_i is given as,

$$P_i = \left(\frac{n}{m}\right)\left(\frac{n-1}{m-1}\right)...\left(\frac{n-i+2}{m-i+2}\right)$$

Since $n < m$,

$$P_i \leq \left(\frac{n}{m}\right)^{i-1}$$

Let the indicator random variable X_i represent whether ith probe occurred. Then,

$$X_i = \begin{cases} 1 & if\ ith\ probe\ occured \\ 0 & else \end{cases}$$

$$P(X_i = 1) = P_i \leq \left(\frac{n}{m}\right)^{i-1} = \alpha^{i-1}$$

$$E[X_i] = 1 \times P(X_i = 1) <= \alpha^{i-1}$$

Let the random variable X represent number of probes. Then,

$$X = X_1 + X_2 + X_3 + ... + X_i + ...$$

Thus, the expected number of probes is given by,

$$E[X] = E[X_1] + E[X_2] + E[X_3] + ... + E[X_i] + ...$$

$$E[X] \leq a^0 + a^1 + a^2 + ... + a^{i-1} + ...$$

$$E[X] \leq \frac{1}{1-\alpha}$$

Hence. the expected number of probes during unsuccessful search is at most $1/(1-\alpha)$

If hash table is half full, $\alpha = 0.5$, at most 2 probes.
If hash table is 90% full, $\alpha = 0.9$, at most 10 probes.

Assuming simple uniform hashing, when open-addressing is used for collision resolution, a successful search for $\alpha < 1$ takes at most

$$\frac{1}{a} \times log\Big(\frac{1}{1-\alpha}\Big)$$

Consider searching for an item x whose key is k in a hash table containing n items in m slots, with $n < m$.
First point to realize is that the sequence of probes required for searching key k is the same as the sequence of probes used when inserting the key. Since inserting a key requires an unsuccessful search followed by placing it on the first empty slot found, expected number of probes is at most $1/(1-\alpha)$.

Thus if k was inserted on hash table when it contained i number of items, the expected number of probes to search key k is at most

$$\frac{1}{1-i/m} = \frac{m}{m-i}$$

The above expected value is for a particular $i+1$ item of the hash table.
Averaging over all n, the expected number of probes for any item is at most

$$\frac{1}{n} \times \sum_{i=0}^{n-1} \frac{m}{m-i}$$

Using method of substitution $m - i = y$, we get

$$\frac{m}{n} \times \sum_{y=m+n-1}^{m} \frac{1}{y}$$

Bounding the sum using integral,

$$\leq \frac{1}{\alpha} \times \int_{m+n}^{m} \frac{1}{y} dy$$

$$\leq \frac{1}{\alpha} \times log\Big(\frac{m}{m-n}\Big)$$

Thus. the expected number of probes for a successful search is at most

$$\frac{1}{\alpha} \times log\Big(\frac{1}{1-\alpha}\Big)$$

If hash table is half full, $a = 0.5$, at most 1.387 probes
If hash table is 90% full, $a = 0.9$, at most 2.559 probes

Universal Hashing

Consider a family of hash functions H. The family is said to be $universal$ if it has the following property:

For any two distinct keys x and y, exactly $|H|/m$ of all the hash functions in H maps x and y to the same slot of the hash table containing m slots.

Equivalently,

If we randomly choose a hash function from all possible $|H|$ hash functions of family H, the probability that there will be a collision for two distinct keys x and y is $1/m$.

The collision probability of mapping two distinct keys x and y uniformly at random is also $1/m$. Thus in expectation, universal hashing is a perfect hashing technique.

Example of a universal hash function.

Suppose we need to store 250 IPv4 addresses in a hash table. Each address x is represented as a quadruple $x = (x_1, x_2, x_3, x_4)$ and $x_i \in [0, 255]$. Let size of the hash table m be 257 which is a prime larger than 250.

For any four coefficients $(a_1, a_2, a_3, a_4) \in [0, m-1]$, let h_a be the hash function defined as,

$$h_a(x_1, x_2, x_3, x_4) = \sum_{i=1}^{4} a_i \times x_i \quad \% \, m$$

A family of hash functions H is then defined as

$$H = \{h_a : a \in [0, m-1]^4\}$$

A hash function can be randomly drawn from H by simply picking four coefficients a_1, a_2, a_3, a_4 at random.
In what follows, we show that H indeed is universal.

Let $x = (x_1, x_2, x_3, x_4)$ and $y = (y_1, y_2, y_3, y_4)$ be distinct keys. Without loss of generality, let them differ in their x_4 and y_4. i.e $x_4 \neq y_4$

Let h_a be the randomly chosen hash function such that:

$$h_a(x_1, x_2, x_3, x_4) = h_a(y_1, y_2, y_3, y_4)$$

Using the definition of h_a,

$$(a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4) = (a_1y_1 + a_2y_2 + a_3y_3 + a_4y_4) \quad \% \, m$$

We wish to calculate the probability that above equation holds.
Suppose we first chose a_1, a_2 and a_3. Then,

$$a_1(x_1 - y_1) + a_2(x_2 - y_2) + a_3(x_3 - y_3) = a_4(y_4 - x_4) \quad \% \, m$$

Since left-hand side of the equation is a constant, we call it c

$$c = a_4(y_4 - x_4) \quad \% \, m$$

$$a_4 = (c(y_4 - x_4)^{-1}) \quad \% \, m$$

Since $x_4 \neq y_4$ and m is prime, $c/(y_4 - x_4) \% \, m$ is unique. Therefore a_4 must be exactly $(c(y_4 - x_4)^{-1}) \% m$, out of its m possible values. Hence,

$$P(h_a(x_1, x_2, x_3, x_4) = h_a(y_1, y_2, y_3, y_4)) = 1/m$$