Hashing
=======

Hashing is a technique for storing and retrieving information as quickly as possible.

Why?

Balanced Binary Trees support insertion, deletion, search with O(log n) time (Average)

Hashing takes this to O(1)

==========

Which technique you would resort to address the following problem?

Given a string, find whether each of its character(s) are repeated or not.

E.g., ullu. (u repeated, l also repeated)

==========
Sample solution
---

1 Typically, u may setup an array of size = # of all possible characters (ascii 256 chars and array size 256) to store # of occurrences corresponding to each char. Initialize the counters to 0.

2 For each of the char given in the string s, increase corresponding counter in the step 1 array

3. For each of the char in the string s, compare the value of the # of counters with 1, if > 1 , repeated else unique.

What is the time complexity and space complexity for the above algorithm?

Suppose, instead of chars we need to check numbers for repeated occurrences? Universal set of numbers has infinite size !!

Need of mapping of certain kind

Key and Value

==========

Components of Hashing
==========

- Hash Table
- Hash function
- Collisions
- Collision resolution technique(s)

==========

Hash table

- Generalization of the array
- store element whose key is k at position k of the array

Thus, given a key k, we find element whose key is k by just reading kth position of array (Direct addressing)

Link up with the example given earlier.. (Array allocated with one position for every possible key)

What if we have less number of locations (finite) and more possible keys?

Hash Function

A function to transform the key into index
Ideally this function should map each possible key to a unique slot index
(Difficult to achieve)

Please note::
key in the first example was ascii char itself, value was # of its occurrences.

Given a collection of elements, a hash function that maps each item into a unique slot is referred to as a perfect hash function.

So if we know the elements and the collection (which is not changing!) then it is possible to construct a perfect hash function.

What if we increase the size of the hash table to make it LARGE?
Then for large number of keys, we can store values!

Above approach not feasible when we are dealing with LARGE size.

Case of aadhar card number. # of digits in aadhar card?
How many slots required?

Collision:: is a situation that occurs when two distinct pieces of data result into same value

If H is hash function, then

H(x) = H(y) and x != y

Objective is to minimize # of collisions

Folding method to construct a hash function

If we are dealing with telephone numbers as elements. # of digits in the telephone number is 10...LARGE

Another trick could be

> Lets say array size of 1,000 is adequate,
> Then one needs to divide the 10-digit number into three groups of three digits and one group of one-digit, add up these number and % with the size of array.
>
> Number 7709050248, i.e., 770+905+024+8 = 1707 and
> 1707 % 1000 = 707

So we can associate array index 707 to deal with the # 7709050248

Take another 10-digit number, calculate its hash value.

If you find another 10-digit number with same hash value, we have a collision.

Load factor

The load factor of a non-empty hash table is the number of items stored in it divided by the size of the table.

Load factor = (# of elements in the hash table)/Size of the hash table.

This parameter is used to determine efficiency of hashing function as well as used when to rehash!

Characteristics of good hash function

- Minimum collision
- Easy and quick to compute
- Distribute key values evenly in the hash table
- Use all the information given in the key

Collision resolution technique
==========

What is collision resolution?

Collision resolution is a process of finding alternate location where our value can be found.

Techniques to be discussed
==========
Chained Hash Table
Linear Probing
Quadratic Probing
Double Hashing

A) Chained Hash Table::

Linked list with hash table

When two or more records hash to the same location, these records are constituted into a singly linked list called as a chain.

Universe of possible keys
Used keys
Hash table
Singly Linked list

B) Open Addressing ::

All keys are stored in the hash table itself.
Also called as closed hashing.

Collision is resolved by probing.

- Linear Probing
- Quadratic Probing
- Double Hashing

Linear Probing

Search the hash table sequentially starting from original hash location. If the location n is occupied, we go to the next location using

rehash(key) = (n+1)%tablesize

Clustering issue crops up
Small clusters may come closer to each other to form large clusters.
This causes long probe searches and decreases overall efficiency.

Step size of more than 1 is possible and should be relatively prime to the size of the table.
If the table size is prime itself then step size could be any number!!

Quadratic Probing

In case of quadratic probing, the rehash function takes the following form

rehash(key) = (n + (k^2)) % tablesize

Table size 11 elements
hash(key) = key mod 11

Insert keys: 31, 19, 2, 13, 25, 24, 21, 9

0	1	2	3	4	5	6	7	8	9	10
		2	13	25	5	24	9	19	31	21

Double Hashing

Benefit

Reduces clustering in a better way compared to the earlier 2 approaches of probing

Conditions for Double Hashing functions

If h1 represents first hashing, h2 represents second hashing function then, we need to ensure

h2(key) != 0 and h2 != h1

Collision Resolution

First probe using h1(key)

If location occupied(means collision),
probe h1(key)+h2(key), still collision, use h1(key)+2*h2(key)

E.g.:

Table size: 11
h1(key) = key mod 11
h2(key) = 7 - (key mod 7)

Insert keys: 58, 14, 91, 25

For 91, 91 mod 11 = 3 (location 3 occupied with 58
applying h1(key)+h2(key)
3 + (7-(91 mod 7)) = 3 + (7-0) = 10 which is occupied with 14

so apply h1(key)+2*h2(key)
3+2*(7) = 17 with mod 11 its 6.

Result::

0	1	2	3	4	5	6	7	8	9	10
			58	25		91			25	14

==========

Comparisons

Linear Probing Vs Double Hashing
==========
cost of computing and load factor
double hashing takes more time to compare two hash functions for long key