Your Ultimate Guide To Landing Top AI roles

DECODE AiML

Hash Table

→ The main reason behind using Data structure is to store data either to give input to algorithm or to store the output of an algorithm.

→ Common operations in Data structure

① Insertion ② Search ③ Deletion

→ Let's compare the Search time of data structures

- Unsorted Array — $O(n)$
- Sorted Array — $O(\log n)$
- Linked List — $O(n)$
- Binary Tree — $O(n)$
- Binary Search Tree — $O(n)$
- Balanced BST (AVL) — $O(\log n)$
- Heap — $O(n)$

$\boxed{\text{The best search time on comparing all the algorithms is } O(\log n)}$

★ Can we have a data structure whose search is $< O(\log n)$?

$\boxed{\text{Hash Table is a data structure on which search can be done in } O(1) \text{ time}}$

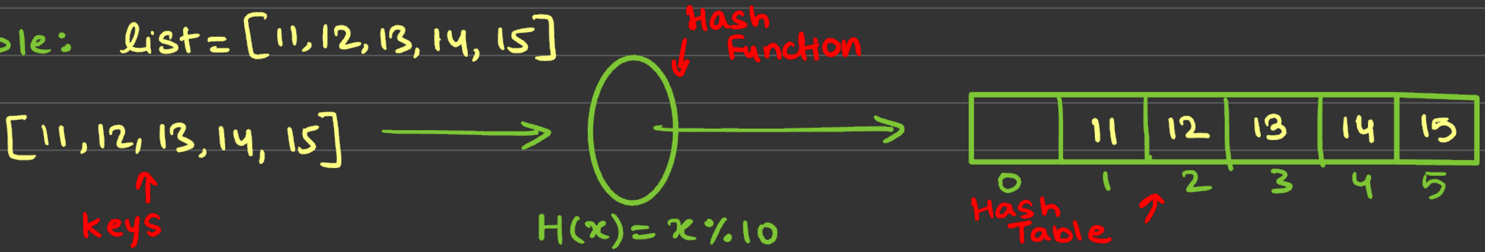
→ Hash Table is a data structure that stores [key → value] pairs and gives (on Average/Amortized) O(1) time for insert/search/delete.

→ It Computes an index from the key using a hash function and Stores the pair in an array (or bucket)

→ Hash table operates on the Concept of Hashing

→ Hashing generates a fixed size output from an input of Variable size using a mathematical formula Called Hash Function

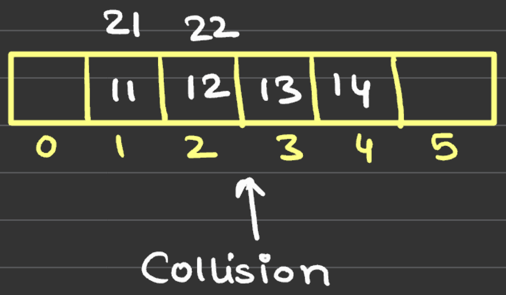
Example: list = [11, 12, 13, 14, 15]



Q. what if list = [11, 12, 13, 22, 21, 14]

→ In that case

H(11) = 1
H(12) = 2
H(13) = 3
H(14) = 4
H(21) = 1
H(22) = 2

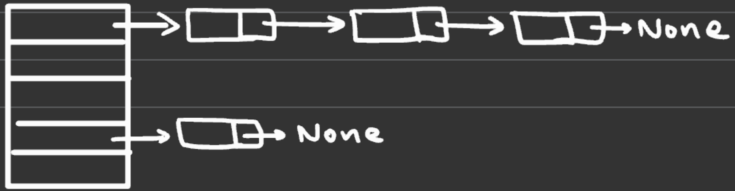


→ More than 1 element mapping to same index in hashtable. This is Collision.

* **Solution of Collision**

① Coming up with better hash function

② Chaining : use linked list



③ Open Addressing: dynamic hash function

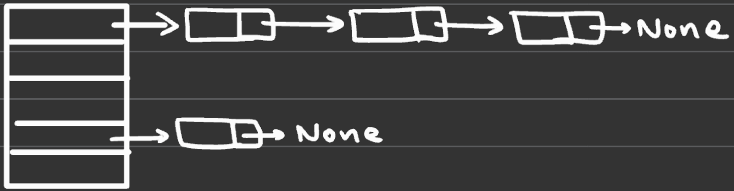
a. Linear Probing

b. Quadratic Probing

c. Double Hashing

* Hashing with Chaining

→ In Chaining, we used Linked List to handle Collision.



→ Chaining is good if we need deletion operation as well.

→ Time Complexity. (worst scenario)

① Insertion = $O(1)$

② Search = $O(n)$ ← All key in one bucket.

③ Deletion = $O(n)$

→ using good Hash Function and resizing to keep α Constant

$\underset{\text{Uniform distributed}}{\uparrow}$ good; $\underset{\text{dynamic Array}}{\uparrow}$ resizing

$$\boxed{\alpha = \frac{n}{m}}$$

Load Factor (α)

n ← no of elements to insert

m ← Hash table size

Amortized Search Time $= O(1+\alpha)$

$= O(1)$

→ Extra Space for pointers.

Average Case:- If the universe is nice to me most of the time.

Amortized Case: Even If universe is Evil, my long term average Stays low.

* When to use Amortized Analysis?

→ Amortized analysis needs to be used when a data structure have

① Most operation → very Cheap - $O(1)$

② Occasional operations → Very expensive - $O(n)$

Example :- dynamic Array $\boxed{\text{Lecture - 2.4}}$

* Hashing with Open Addressing

→ Open addressing mean we are going to insert all element inside the table only.

→ we can't insert more elements than size of table

Load Factor $\boxed{0 \leq \alpha \leq 1}$

→ Open addressing is good when we don't need deletion.
↳ Complicated.

→ Collision is resolved by reapplying Hash function with Some changes ⟶ called Probing.

→ Primary Clustering arise in Linear Probing and Secondary Clustering arise in Quadratic Probing.

Time Complexity

→ Time Complexity (worst scenario) → Poor hash function
→ Table almost full ($\alpha \to 1$)
→ Primary clustering
→ Secondary clustering.

1. Insert → $O(n)$
2. Search → $O(n)$
3. Delete → $O(n)$

→ Amortized Time Complexity.

↳ Insert / Delete / Search

↳ $\boxed{T(n) = O(1/(1-\alpha))}$ ← Expected no of probes is $1/(1-\alpha)$

↳ If α is Constant ($\alpha \leq 0.75$)

↳ $T(n) = O(1)$

Ex:- Apply Linear probing.

GATE-08

keys: 12, 18, 13, 2, 3, 23, 5 and 15

Hash Table Size = 10 (m)

$h(k) = k \% 10$

$\rightarrow \boxed{h'(k, i) = (k + i) \% 10}$

insert(12) →

0	1	2	3	4	5	6	7	8	9
		12							

insert(18) →

0	1	2	3	4	5	6	7	8	9
		12						18	

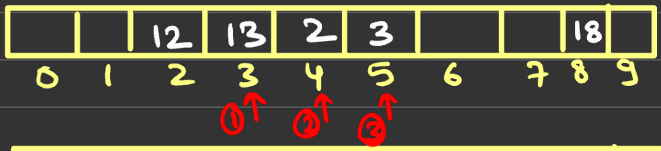
insert(13) →

0	1	2	3	4	5	6	7	8	9
		12	13					18	

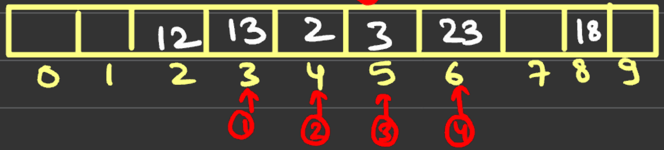
insert(2) →

0	1	2	3	4	5	6	7	8	9
		12	13	2			18		
		↑ $h(2)$	↑ $h'(2,1)$	↑ $h'(2,2)$					

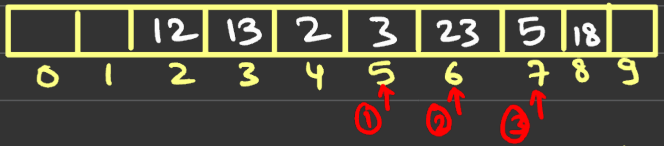
insert(3) →



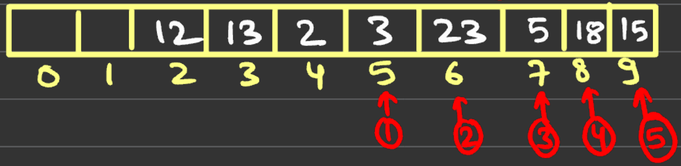
insert(23) →



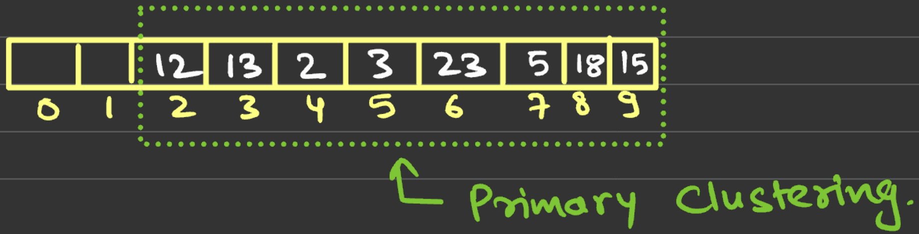
insert(5) →



insert(15) →



→ Primary Clustering



→ Quadratic Probing

Linear Probing → $h'(k,i) = (h(k)+i) \% m$

$$\Downarrow$$

$$\boxed{h'(k,i) = (h(k) + c_1 \cdot i + c_2 \cdot i^2) \% m}$$

↳ Disadvantage:

① Secondary Clustering. ← same probe sequence.

② may not probe all table slots

→ Double Hashing

$$\boxed{h'(k,i) = (h_1(k) + i \times h_2(k)) \% m}$$

→ No Primary or Secondary Clustering.

→ Double hashing generally performs better, especially at higher load factor (α)

Like ▶ Subscribe