Hashing: Suppose we wanted to have dictionary then common operations are going to be

- Insert
- Search
- Delete

(*) Hash Tables are an effective and efficient data structure for such dictionary operations.

⟹ Searching in Hash Table

⟹ $\Theta(n)$ worst case

⟹ $\Theta(1)$ Average/expected case.

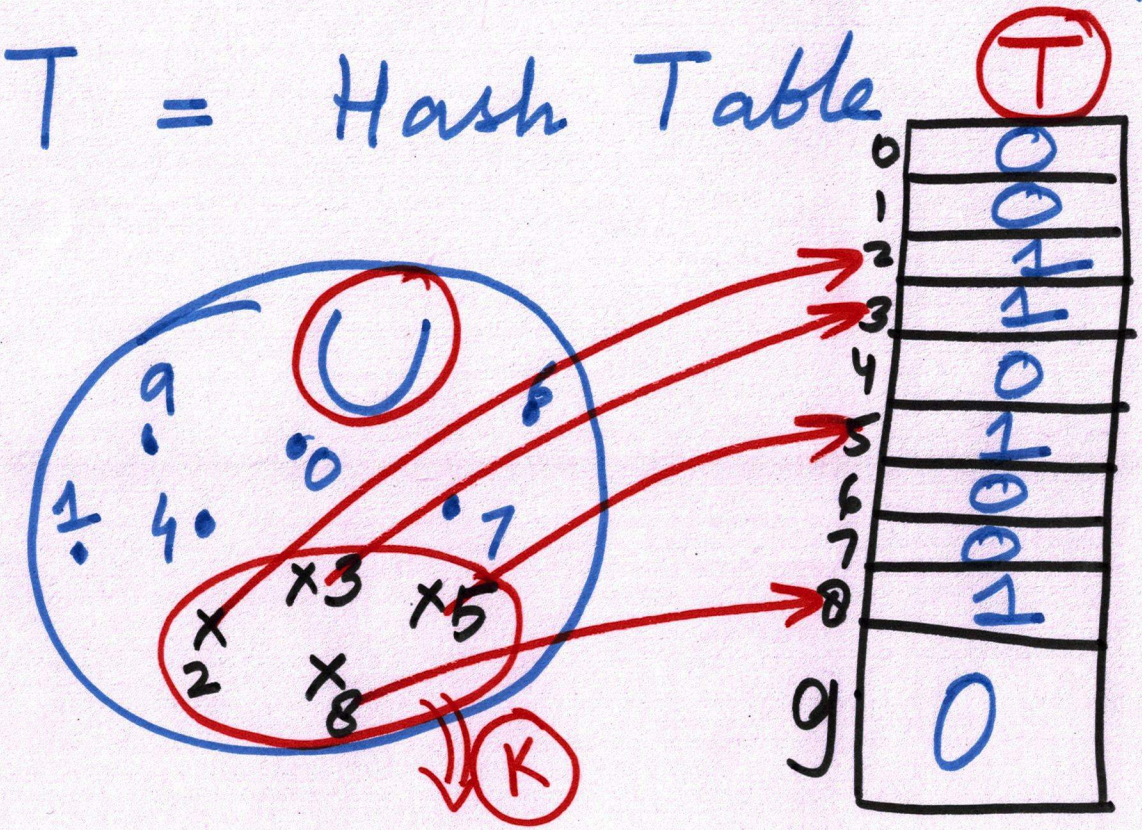
Generalization of simple ordinary array.

Direct Address Table

U = Universe of all possible keys
Let it be $\{0, 1, 2, ----- 9\}$

K = Actual set of available Keys
Let it be $\{2, 3, 5, 8\}$

T = Hash Table



If stored like this then all dictionary operations can be done in O(1) time. This is called Direct addressing

Direct-Address

- Search $(T, k) \rightsquigarrow$ Return $(T(k))$
- Insert $(T, x) \rightsquigarrow T[k] \leftarrow 1$
- Delete $(T, x) \rightsquigarrow T[k] \leftarrow$ NIL (or 0)

Problems with Direct Addressing

if U is very large $\rightarrow |U| \Rightarrow$ impractical

and $|K|$ is practically very small

$$|K| << |U|$$

Most allocated space for (T) got wasted.

Hash table must require much less storage space than direct address table.

↳ Infact storage requirement can be reduced to $\Theta(K)$ and still searching requires $O(1)$ time.

⇓ Catch is that this time it will be average case time, instead of worst case time.

Indirect addressing

Previously Key(K) is stored in slot(K).

Now in Hash Table:

Element with Key(K) will be stored in slot $h(K)$ where

A hash function (h) is used to compute the slot from the key (K).

$$h : U \longrightarrow \{0, 1, 2, --- m-1\}$$

$$K \xrightarrow[\text{to slot } h(K)]{\text{hashes}} h(K)$$

Basically function $h \equiv$ reduces the range of array indicies that are needed to be handled (Size of T)

Collision : If 2 Keys hashes to the same Slot then collision occurs.

There are Several ways to handle Collision

Collision resolution → (a) By Chaining

$h(K_2) = h(K_5)$

↳ Collision

(b) Open addressing
- Linear Probing
- Quadratic Probing

(A) Collision Resolution by chaining

- Put all elements that hashes to the same slot in a linked list.
- Any slot j, contains a pointer to the head of the linked list of all stored elements that hash to j.
- If there are NO such element, slot j contains NIL.

(*) Chained Hash Insert (T, x)

insert (x) at the head of list $T[h(key[x])]$