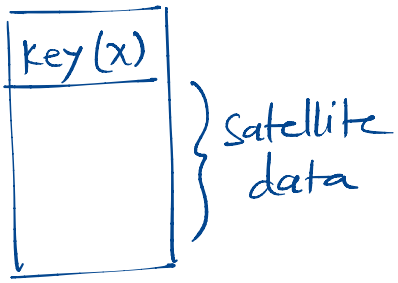
Hashing

Hash Table: A generalization of the notion of an ordinary array

→ It's a dynamic set supporting operations such as: Insert, Delete, search ...

Record x: 

Dynamic set operations on set S:

- Insert (S, x): $S \leftarrow S \cup \{x\}$
- Delete (S, x): $S \leftarrow S - \{x\}$
- Search (S, k): return record
  - x s.t. $key(x) = k$
  - or
  - NIL if no such x

Direct Access Table — Use an array to store records
— Index by key

→ This works well when keys are drawn from a small distribution

- Suppose keys are drawn from the universal set $U \subseteq \{0, 1, \ldots, m-1\}$, and keys are distinct.

Then setup an array $T[0, \ldots, m-1]$ to represent dynamic set S, such that

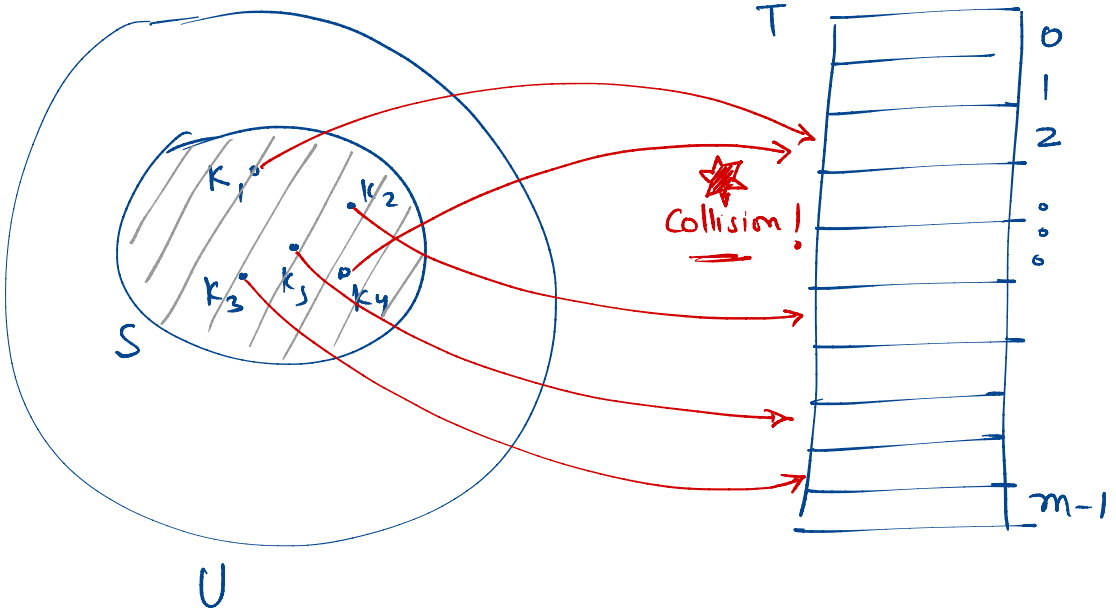
$$T[k] = \begin{cases} x, & \text{if } x \in S \text{ and } key[x] = k \\ NIL, & \text{otherwise} \end{cases}$$

Problem: If range of keys is large then it is impractical to use an array.

e.g. - Storing 1000 10-digit phone numbers.

- 64-bit numbers! $\sim 18 \times 10^8$
- Character strings

Hashing: We want to keep the table size small while still retaining good properties of an array.



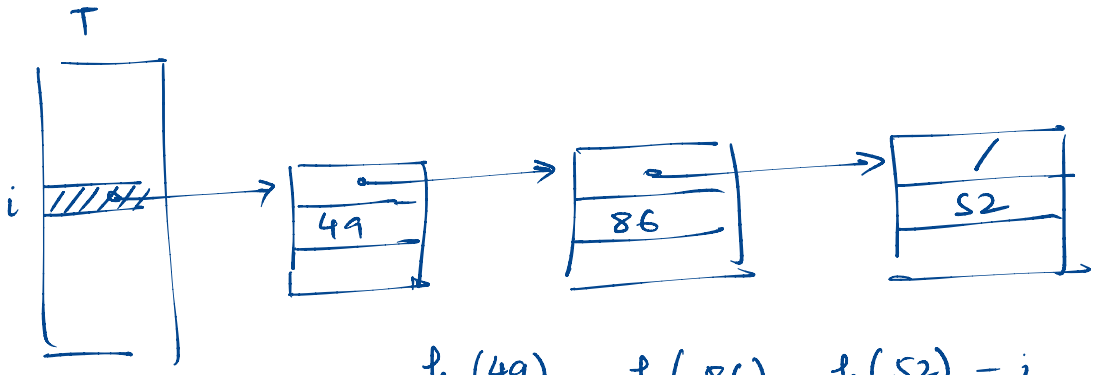
Collision: A collision occurs when a record to be inserted maps to an already occupied slot in T.

Addressing → Open addressing
→ Closed addressing (chaining)

→ Example of a hash function to map a key k into a table of size m : $h(k) = k \bmod m$

Chaining : resolve collisions by chaining

↳ link records in the same slot using a list.



$h(49) = h(86) = h(52) = i$

Analysis of hashing :

Worst Case : - Every key hashes to the same slot

- Access time : $\Theta(n)$ if $|S| = n$

Average Case : Assume simple uniform hashing

↳ distribute keys randomly to slots.

↳ Each key $k \in S$ is equally likely to be hashed into any slot of the table T, independent of where other keys are hashed.

Let n be the number of keys in S

Let m " " " " slots in T

Define load factor of T as :

$\alpha = n/m$ = avg. no. of keys / slot

- The expected time for an unsuccessful search for a record with a given key is

$= \Theta(1 + \alpha)$

expected cost of searching a list

Const. work to hash a key

So, Expected search time is $\Theta(1)$ if $\alpha = O(1)$

or

in other words, if $n = O(m)$.

- A successful search also has the same asymptotic complexity.