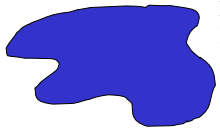
Hash Function Needs to Be Good!

An ideal hash function:
- Is fast to compute
- "Rarely" hashes two "used" keys to the same index
  - Often impossible in theory; easy in practice
  - Will handle *collisions* a bit later

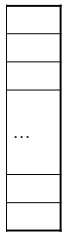


key space (e.g., integers, strings)

hash function:
h(key) → int

int mod Tablesize → index

hash table



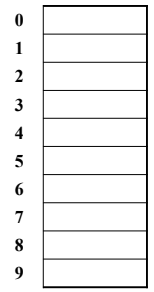
10

Hashing integers (try it out)

key space = integers

Simple hash function:
- Client: `h(x) = x`
- Library `g(x) = h(x) % TableSize`
- Fairly fast and natural

Example:
- TableSize = 10
- Insert **7, 18, 41, 34, 10**
- (As usual, ignoring corresponding data)



14

What if the key is not an int?

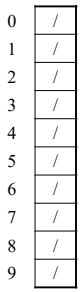
- If keys aren't `int`s, the **client** must convert to an `int`
  - Trade-off: speed and distinct keys hashing to distinct `int`s
- Common and important example: Strings
  - Key space K = $s_0 s_1 s_2 \ldots s_{m-1}$
    - where s_i are chars: $s_i \in [0,256]$
  - Some choices: Which avoid collisions best? What strings would collide?

1. h(K) = s_0
2. h(K) = $\left(\sum_{i=0}^{m-1} s_i \right)$
3. h(K) = $\left(\sum_{i=0}^{m-1} s_i \cdot 37^i \right)$

Then on the **library** side we typically mod by Tablesize to find index into the table

18

Separate Chaining



Chaining: All keys that map to the same table location are kept in a list (a.k.a. a "chain" or "bucket")

As easy as it sounds

Example: insert 10, 22, 107, 12, 42 with mod hashing and `TableSize` = 10

25