Hashing

JONAS BURO

MAY 2023

CSC 226 UVIC

Why hash

- Efficiently store, retrieve, and check data (hashtables)
- Ensure data integrity
- Password storage in db’s
- Accelerates other processes (string comparison, substring count)

Hash table

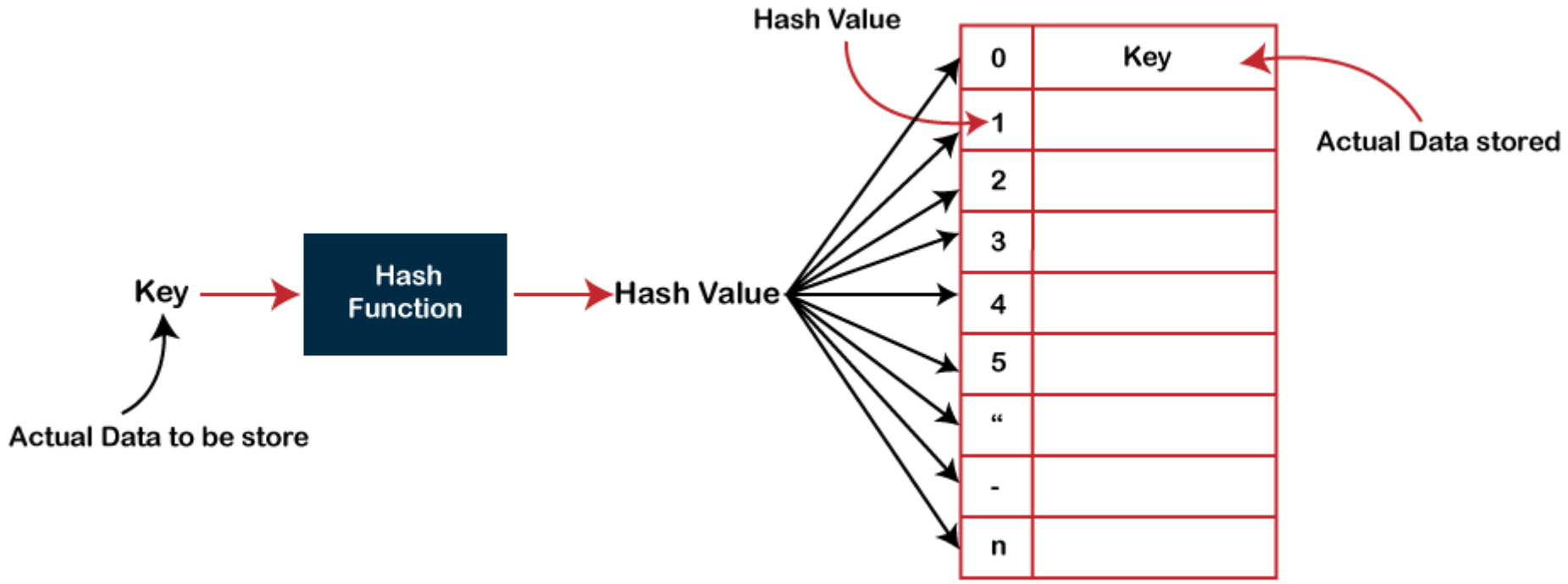

image source https://www.javatpoint.com/hash-table

Hash function

- **INPUT:** object, **OUTPUT:** hashcode (int)
- Should be fast to compute
- Should minimize collisions
- Data dependent

Hash function example

- Polynomial rolling hash function for strings of length *n:*

$$\begin{aligned}\text{hash}(s) &= s[0] + s[1] \cdot p + s[2] \cdot p^2 + \ldots + s[n-1] \cdot p^{n-1} \mod m \\ &= \sum_{i=0}^{n-1} s[i] \cdot p^i \mod m,\end{aligned}$$

- for user defined *p, m*

Collision resolution

- Given function *h,* objects *a,b*:
- If *h(a) = h(b),* then we have a collision
- We must resolve this in order to maintain integrity of the hashtable
- Multiple approaches

Probing

- If the value of *h(U) is already occupied, find an empty slot h(U,i)*
- Linear Probing $h(U, i) = [h(U) + i] \mod N$
- Quadratic Probing $h(U, i) = [h(U) + i^2] \mod N$
- NOTE: this must be taken into consideration not only during insertion into the hashtable, but searching it as well

Other considerations

- Load factor
- Resizing hashtable
- Measuring performance

Codebase

- Small Java hashing exercise
- Main: measures execution time for insertion and search phase
- Program executable in interactive or file-read mode
- TODO: complete method implementations
  - hash
  - insert
  - find