Double Hashing

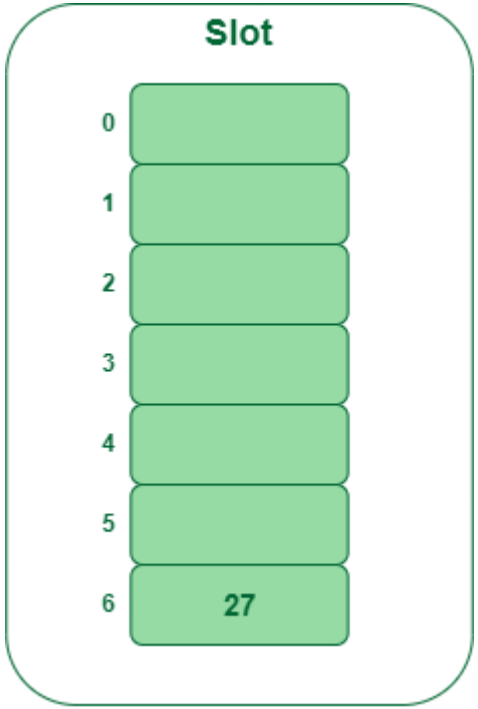
Double Hashing

The intervals that lie between probes are computed by another hash function. Double hashing is a technique that reduces clustering in an optimized way. In this technique, the increments for the probing sequence are computed by using another hash function. We use another hash function hash2(x) and look for the i*hash2(x) slot in the **i^{th}** rotation.

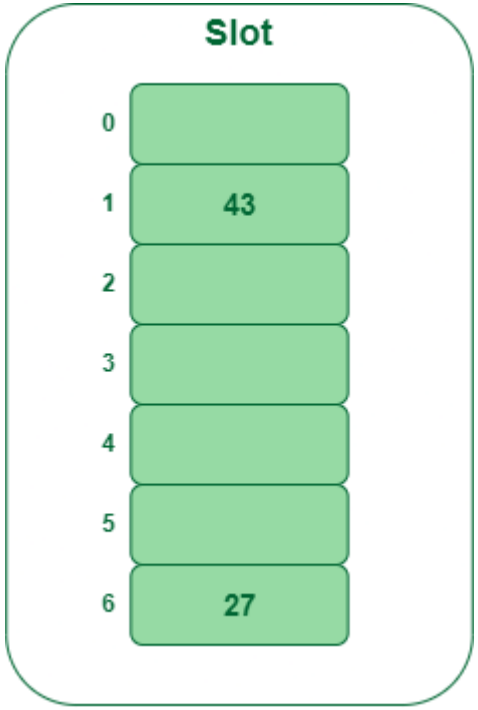
> *let hash(x) be the slot index computed using hash function.*
>
> *If slot hash(x) % S is full, then we try (hash(x) + 1*hash2(x)) % S*
> *If (hash(x) + 1*hash2(x)) % S is also full, then we try (hash(x) + 2*hash2(x)) % S*
> *If (hash(x) + 2*hash2(x)) % S is also full, then we try (hash(x) + 3*hash2(x)) % S*
> .....
> .....

Example: Insert the keys 27, 43, 98, 72 into the Hash Table of size 7. where first hash-function is **h1(k) = k mod 7** and second hash-function is **h2(k) = 1 + (k mod 5)**

- **Step 1:** Insert 27
  - 27 % 7 = 6, location 6 is empty so insert 27 into 6 slot.



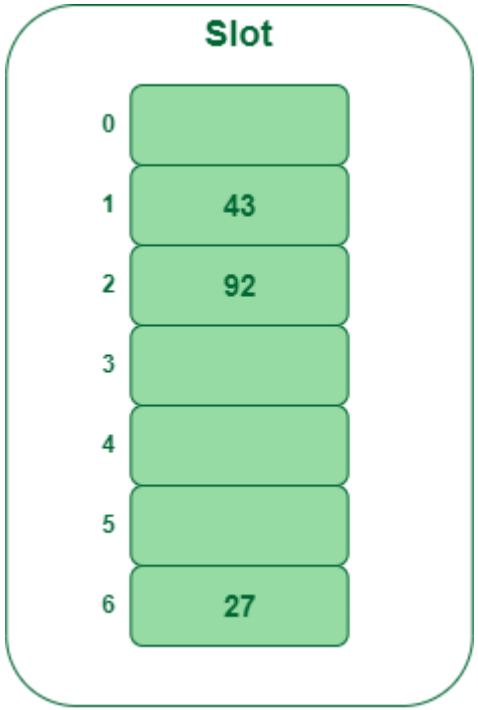
- **Step 2:** Insert 43
  - 43 % 7 = 1, location 1 is empty so insert 43 into 1 slot.



- **Step 3:** Insert 92
  - 92 % 7 = 6, but location 6 is already being occupied and this is a collision
  - So we need to resolve this collision using double hashing.

> h_{new} = [h1(92) + i * (h2(92)] % 7
> = [6 + 1 * (1 + 92 % 5)] % 7
> = 9 % 7
> = 2

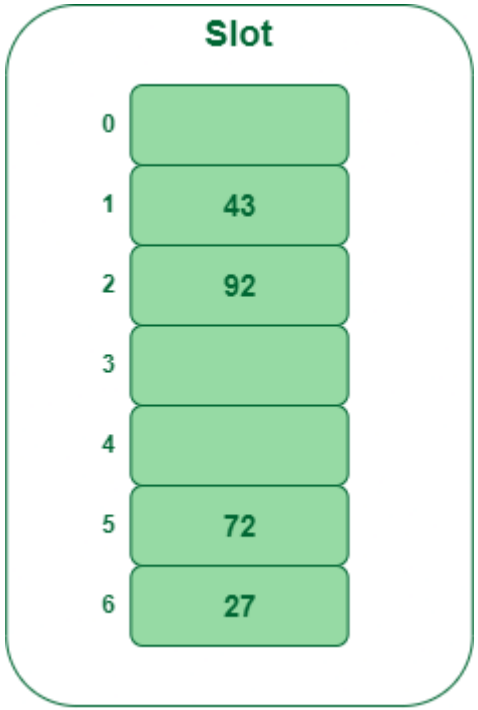
> *Now, as 2 is an empty slot,*
> *so we can insert 92 into 2nd slot.*



- **Step 4:** Insert 72
  - 72 % 7 = 2, but location 2 is already being occupied and this is a collision.

- So we need to resolve this collision using double hashing.

> h_{new} = [h1(72) + i * (h2(72)] % 7
> = [2 + 1 * (1 + 72 % 5)] % 7
> = 5 % 7
> = 5,
>
> Now, as 5 is an empty slot,
> so we can insert 72 into 5th slot.



Dash
All
Articles
Videos
Problems
Quiz
Contest

Performance of Open Addressing:

Cache performance of chaining is not good because when we traverse a Linked List, we are basically jumping from one node to another, all across the computer's memory. For this reason, the CPU cannot cache the nodes which aren't visited yet, this doesn't help us. But with Open Addressing, data isn't spread, so if the CPU detects that a segment of memory is constantly being accessed, it gets cached for quick access.

Like Chaining, the performance of hashing can be evaluated under the assumption that each key is equally likely to be hashed to any slot of the table (simple uniform hashing)

> *m = Number of slots in the hash table*
>
> *n = Number of keys to be inserted in the hash table*
>
> *Load factor α = n/m (< 1)*
>
> *Expected time to search/insert/delete < 1/(1 – α)*
>
> *So Search, Insert and Delete take (1/(1 – α)) time*

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.