DATA STRUCTURES AND ALGORITHMS

HASHING

- A technique to convert a range of key values into a range of indexes of an array
- When it comes to data structures, hashing is a technique used to store and retrieve data in a database
- Hashing is an efficient way to store and retrieve data in O(1), because it avoids the need for comparisons between elements
- It also allows duplicate values to be stored in the same structure without causing collisions
- It is also called the mapping technique as larger values are mapped into smaller ones
- There are many different hashing algorithms, but they all share the same basic principle

BASIC TERMINOLOGIES

- Search Keys
- Hash Tables
- Hash Functions

HASH TABLES

- The hash table is an associative data structure that stores data as associated keys
- An array format is used to store hash table data, where each value is assigned its unique index
- By knowing the index of the desired data, we can access it very quickly
- As a result, inserting and searching data is very fast, regardless of data size
- In a Hash Table, elements are stored in an array, and an index is generated using hashing techniques in a data structure

HASH TABLES

- Hash table is basically an array
- However, in hash table whenever data is inserted, deleted or searched it does not require scanning the whole data
- Hash table insert, delete and search a value with the help of hash function
- The Hash table data structure stores elements in key-value pairs where
    - Key- unique integer that is used for indexing the values
    - Value - data that are associated with keys.

HASH FUNCTION

Several hash functions are described in the literature. Here we describe some of the commonly used hash functions:

- **Mid-Square:** In this method, the hash function, h, is computed by squaring the identifier, and then using the appropriate number of bits from the middle of the square to obtain the bucket address. Because the middle bits of a square usually depend on all the characters, it is expected that different keys will yield different hash addresses with high probability, even if some of the characters are the same.
- **Folding:** In folding, the key X is partitioned into parts such that all the parts, except possibly the last parts, are of equal length. The parts are then added, in some convenient way, to obtain the hash address.
- **Division (Modular arithmetic):** In this method, the key X is converted into an integer. This integer is then divided by the size of the hash table to get the remainder, giving the address of X in HT.

HASH FUNCTION

- Value to be inserted:

  8, 2,10, 0, 11,

- Calculate Hash Function/ hash value

  $h(x) = x$

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0		2						8		10	11		

HASH FUNCTION

- Value to be inserted:

  8, 2, 10, 0, 11, 25

- Calculate Hash Function/ hash value

  $h(x) = x$

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0		2						8		10	11		

HASH FUNCTION (DIVISION METHOD)

- Value to be inserted:

  $24, 52, 91, 67, 48, 83$

- Calculate Hash Function/ hash value

  $h(x) = x \; \% \; size \; of \; table$

0	1	2	3	4	5	6	7	8	9
	91	52	83	24			67	48	

HASH FUNCTION (DIVISION METHOD)

- Value to be inserted:

  8, 66, 9, 57, 20,

- Calculate Hash Function/ hash value

  $h(x) = x \ \% \ size\ of\ table$

0	1	2	3	4	5	6	7	8	9
20						66	57	8	9

HASH FUNCTION (DIVISION METHOD)

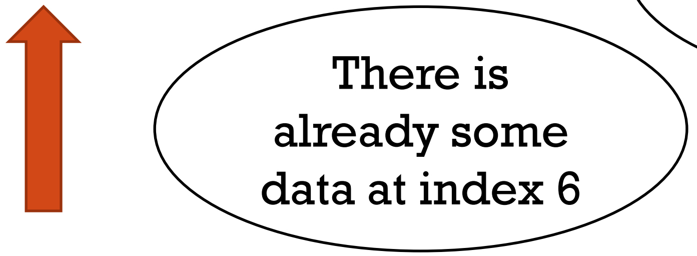
- Value to be inserted:

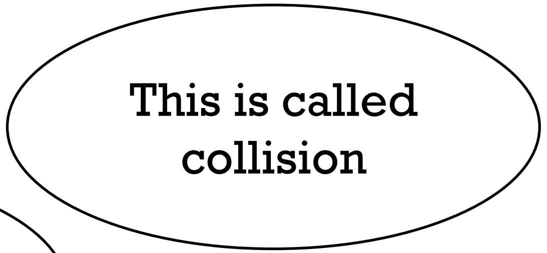
  8, 66, 9, 57, 20, 43, 76

- Calculate Hash Function/ hash value

  $h(x) = x \ \% \ size \ of \ table$

0	1	2	3	4	5	6	7	8	9
20			3			66	57	8	9





COLLISION

- A hash function gets us a small number for a key which is a big integer or string
- There is a possibility that two keys result in the same value
- The situation where a newly inserted key maps to an already occupied slot in the hash table is called collision
- It must be handled using some collision handling technique

COLLISION RESOLUTION TECHNIQUES

- Open Hashing/ Closed Addressing
  - Separate Chaining

- Closed Hashing/ Open Addressing
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

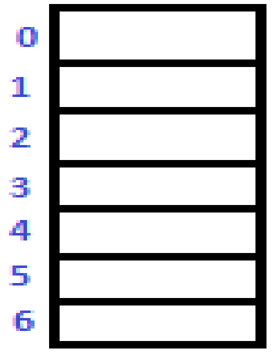
SEPARATE CHAINING

- The idea behind separate chaining is to implement the array as a linked list called a chain
- Separate chaining is one of the most popular and commonly used techniques in order to handle collisions
- So what happens is, when multiple elements are hashed into the same slot index, then these elements are inserted into a singly-linked list which is known as a chain

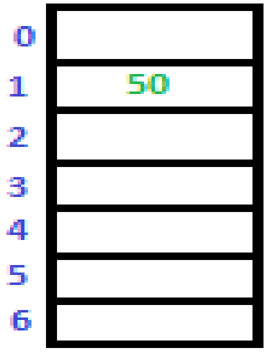
SEPARATE CHAINING

- Let us consider a simple hash function as “key mod 7” and a sequence of keys as 50, 700, 76, 85, 92, 73, 101

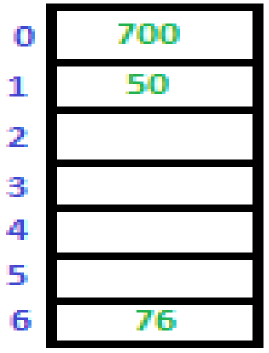
SEPARATE CHAINING



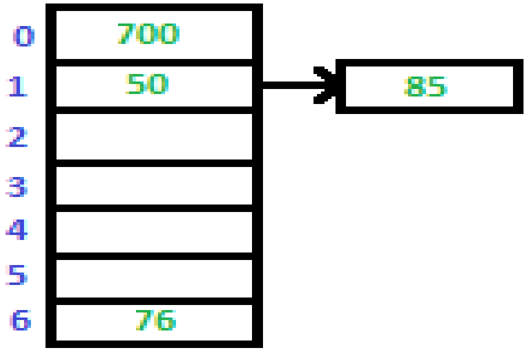
Initial Empty Table



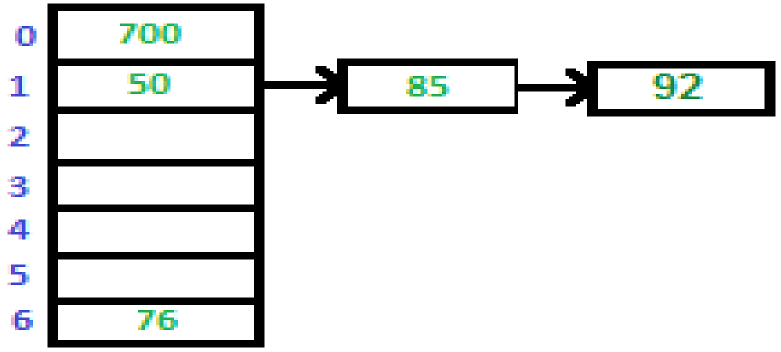
Insert 50



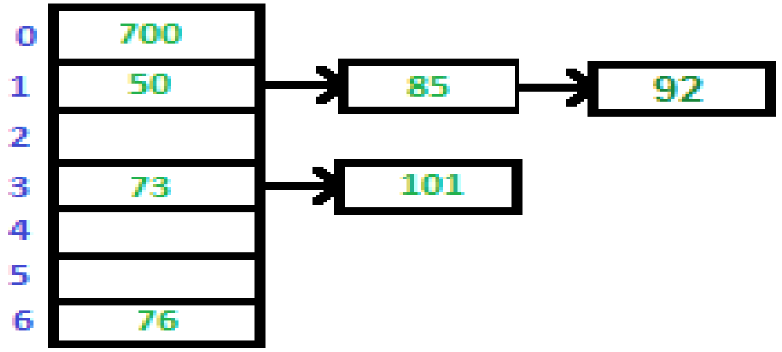
Insert 700 and 76



Insert 85: Collision Occurs, add to chain



Inser 92 Collision Occurs, add to chain



Insert 73 and 101

SEPARATE CHAINING

Advantages

- Simple to implement.
- Hash table never fills up, we can always add more elements to the chain.
- Less sensitive to the hash function or load factors.
- It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

Disadvantages

- The cache performance of chaining is not good as keys are stored using a linked list.
- Wastage of Space (Some Parts of the hash table are never used)
- If the chain becomes long, then search time can become O(n) in the worst case
- Uses extra space for links

OPEN ADDRESSING/ CLOSED HASHING

- Linear Probing
- Quadratic Probing
- Double Hashing

LINEAR PROBING

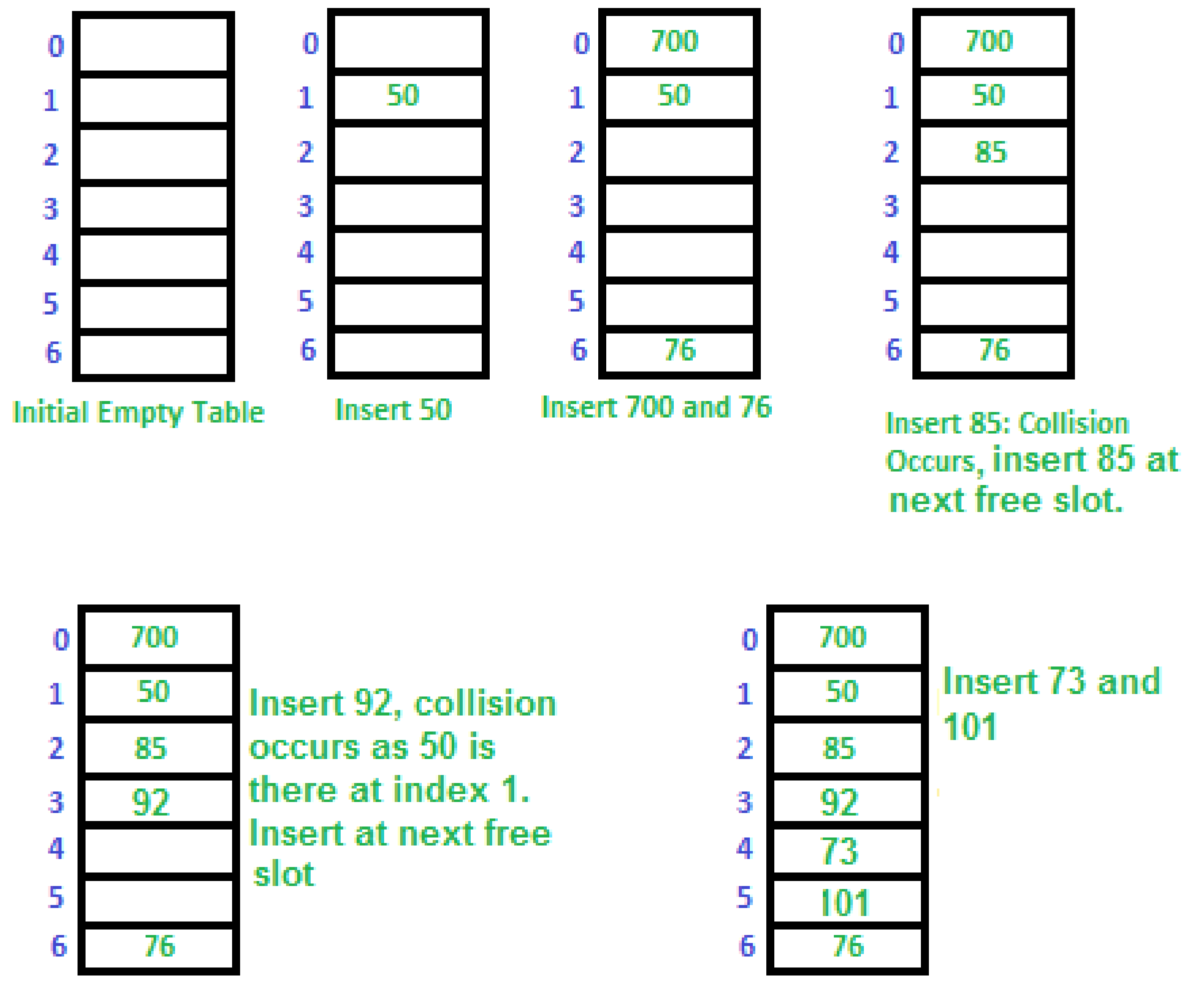
- In linear probing, the hash table is searched sequentially that starts from the original location of the hash.
- Calculate Hash Function/ hash value

$$h(x) = x \,\%\, size\ of\ table$$

- If in case the location that we get is already occupied, then we check for the next location.
- Calculate Hash Function/ hash value again

$$h'(x) = (h(x) + i) \,\%\, size\ of\ table$$

LINEAR PROBING


0 1 2 3 4 5 6
Initial Empty Table
50
Insert 50
700 50 76
Insert 700 and 76
700 50 85 76
Insert 85: Collision Occurs, insert 85 at next free slot.
700 50 85 92 76
Insert 92, collision occurs as 50 is there at index 1. Insert at next free slot
700 50 85 92 73 101 76
Insert 73 and 101

LINEAR PROBING

Advantages

- Overall, linear probing is a simple and efficient method for handling collisions in hash tables
- It can be used in a variety of applications that require efficient storage and retrieval of data
- No extra space is wasted

Disadvantages

- Searching time is O(n)
- Two main challenges for linear probing are:
  - **Primary Clustering:** One of the problems with linear probing is Primary clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search for an element.
  - **Secondary Clustering***:* Secondary clustering is less severe, two records only have the same collision chain (Probe Sequence) if their initial position is the same.

QUADRATIC PROBING

- If you observe carefully, then you will understand that the interval between probes will increase proportionally to the hash value.
- Quadratic probing is a method with the help of which we can solve the problem of clustering that was discussed before.
- This method is also known as the **mid-square** method.
- In this method, we look for the i^2th slot in the ith iteration.
- We always start from the original hash location.
- If only the location is occupied then we check the other slots.

QUADRATIC PROBING

- In quadratic probing, the hash table is searched sequentially that starts from the original location of the hash.
- Calculate Hash Function/ hash value

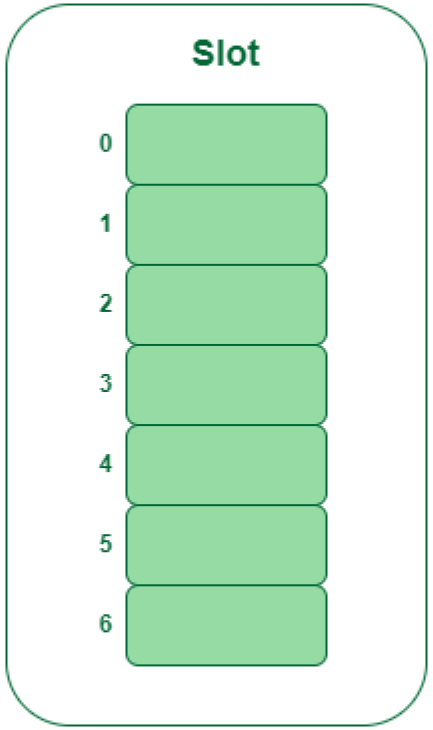
$$h(x) = x \,\%\, size\ of\ table$$

- If in case the location that we get is already occupied, then we check for the next location.
- Calculate Hash Function/ hash value again

$$h'(x) = (h(x) + i^2) \,\%\, size\ of\ table$$

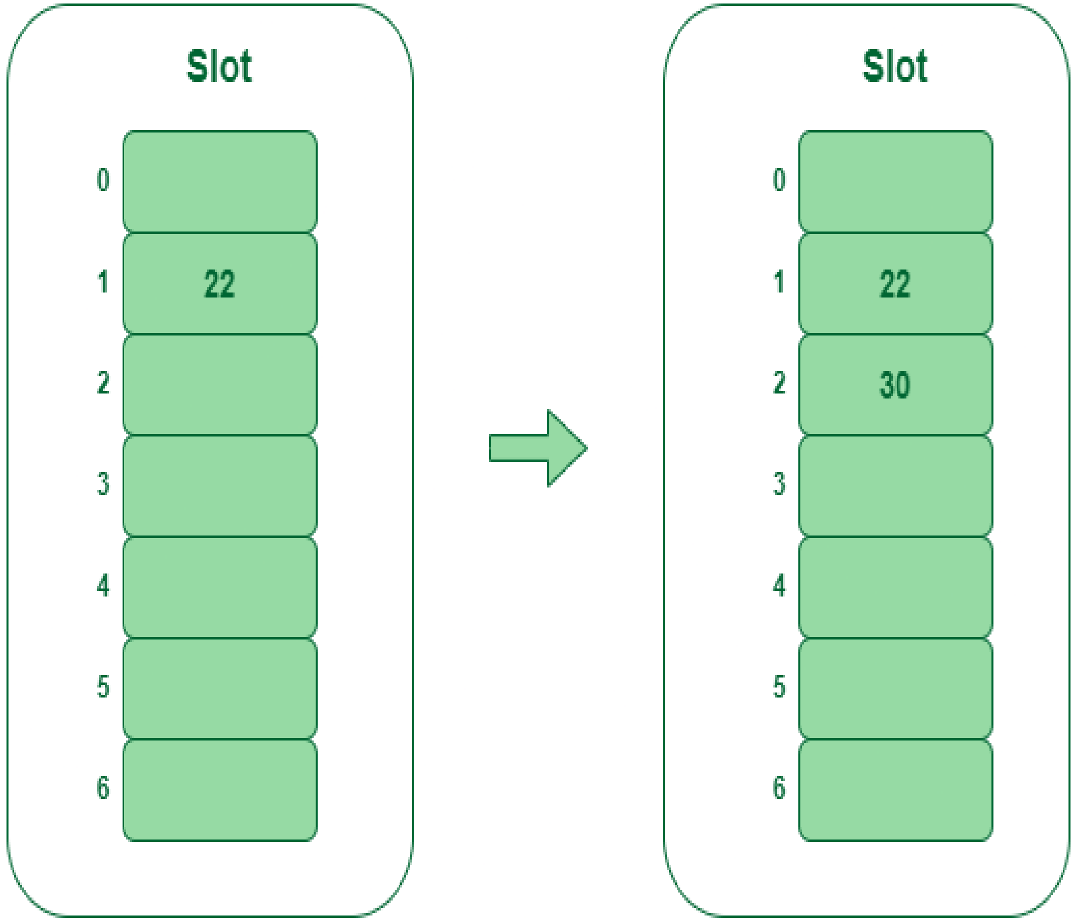
QUADRATIC PROBING

- Let us consider table Size = 7, hash function as Hash(x) = x % 7
- Collision resolution strategy to be f(i) = i^2 . Insert = 22, 30, and 50.
- **Step 1:** Create a table of size 7.



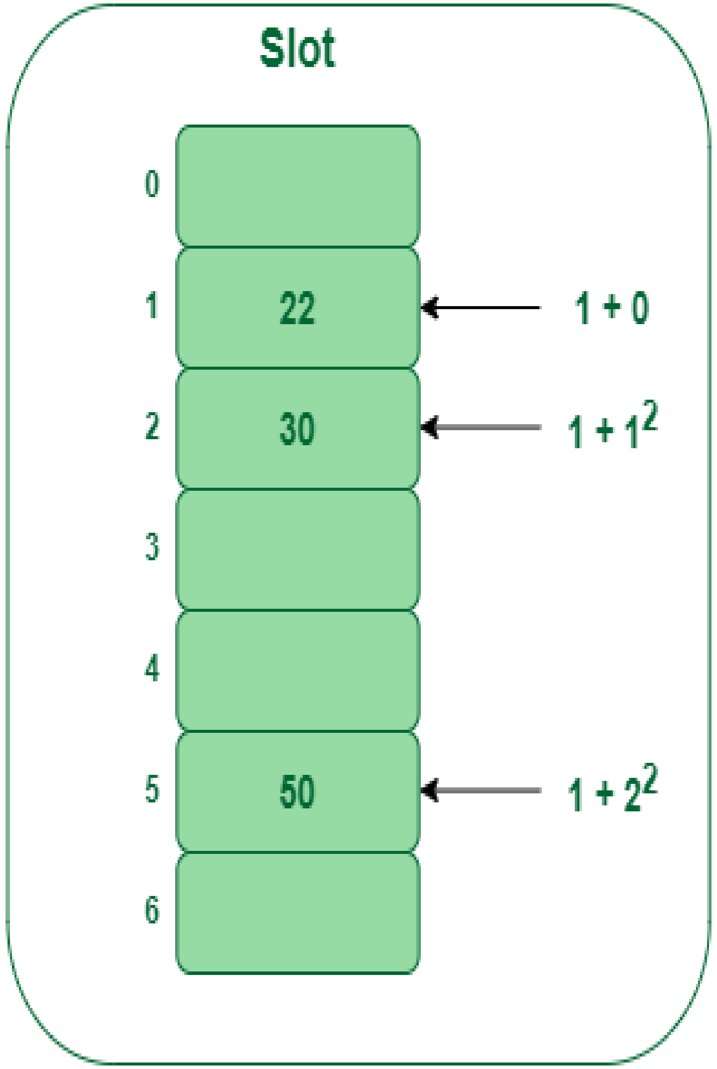
QUADRATIC PROBING

- Hash(22) = 22 % 7 = 1, Since the cell at index 1 is empty, we can easily insert 22 at slot 1.
- Hash(30) = 30 % 7 = 2, Since the cell at index 2 is empty, we can easily insert 30 at slot 2.



QUADRATIC PROBING

- Hash(50) = 50 % 7 = 1
- In our hash table slot 1 is already occupied. So, we will search for slot $1+1^2$, i.e. 1+1 = 2,
- Again slot 2 is found occupied, so we will search for cell $1+2^2$, i.e. 1+4 = 5,
- Now, cell 5 is not occupied so we will place 50 in slot 5.



QUADRATIC PROBING

Advantages

- No extra space is wasted
- Primary clustering issue is resolved

Disadvantages

- Secondary clustering issue
- Search time is O(n)
- No guarantee of finding any slot

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.

DOUBLE HASHING

- Calculate Hash Function/ hash value

$$h1(x) = x \,\%\, size\ of\ table$$

- If in case the location that we get is already occupied, then we check for the next location.
- Calculate Hash Function/ hash value again

$$h'(x) = (h1(x) + i * h2(x)) \,\%\, size\ of\ table$$

where,

DOUBLE HASHING

- If size of table is any prime number,

$$h2(x) = 1 + (x \; mod \; (size \; of \; table - 2))$$

- If size of table is not prime number, then you must select the prime number smaller than the size of table.

$$h2(x) = R \; - x \; mod \; R$$

where R is any prime number smaller than the size of table.

- For example, if the size of table is 13 then

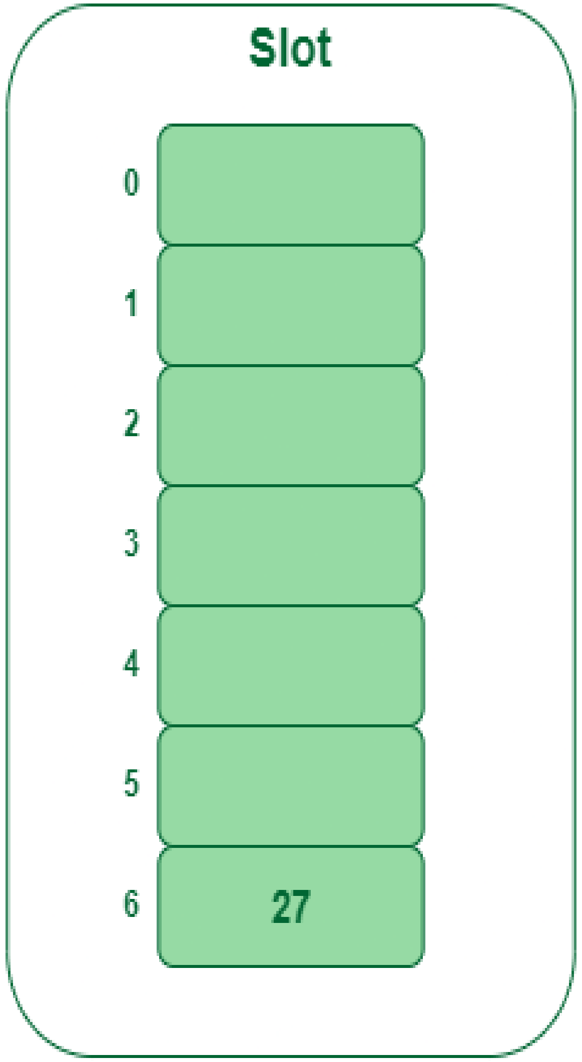
$$h2(x) = 1 + (x \; mod \; (13 - 2))$$
$$h2(x) = 1 + (x \; mod \; (11))$$

- For example, if the size of table is 10 then

$$h2(x) = 7 \; - x \; mod \; 7$$

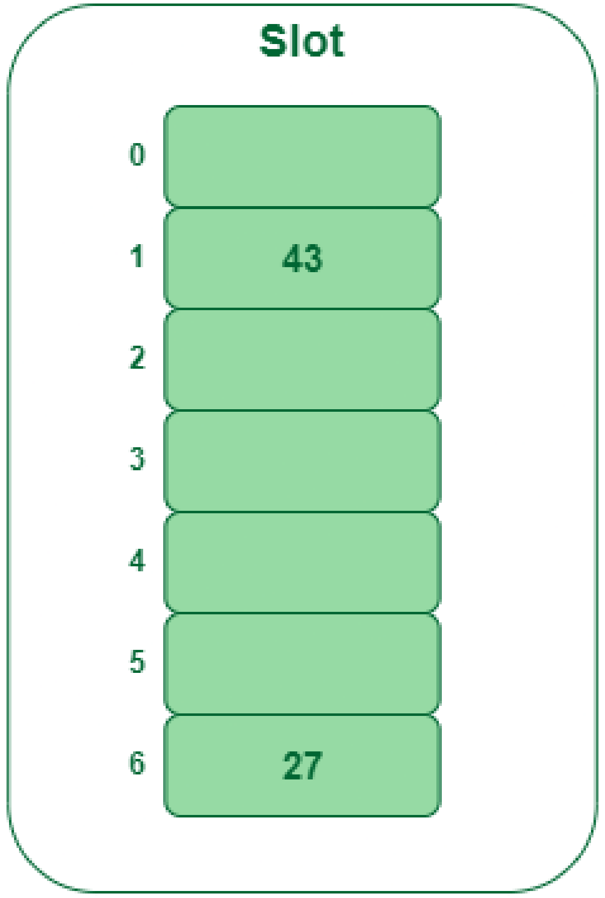
DOUBLE HASHING

- Insert the keys 27, 43, 692, 72 into the Hash Table of size 7.
- First hash-function is h1(x) = x mod 7
- Second hash-function is h2(x) = 1 + (x mod 5)
- Insert 27



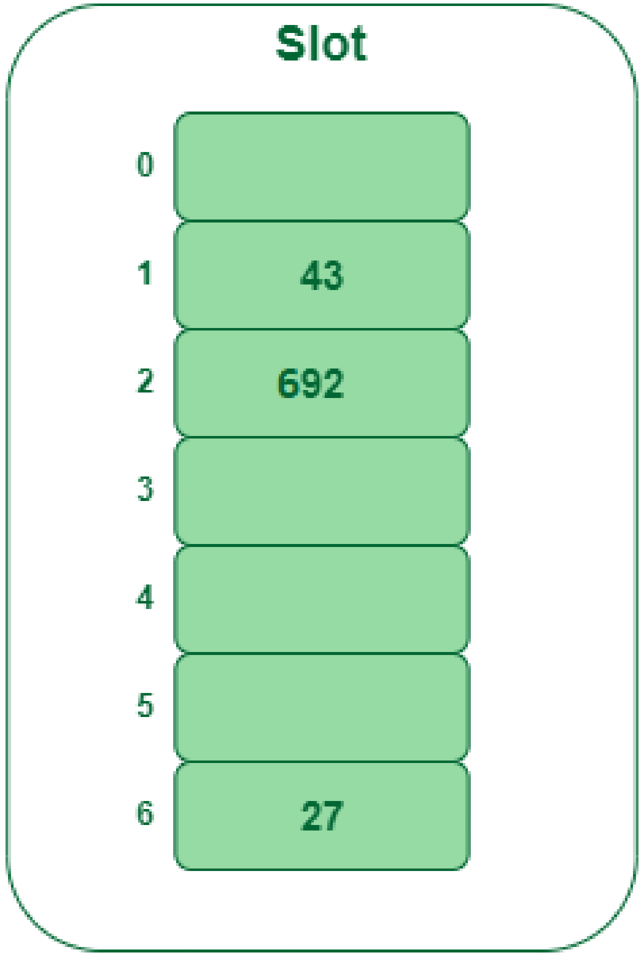
DOUBLE HASHING

- Insert 43 as the desired location is empty



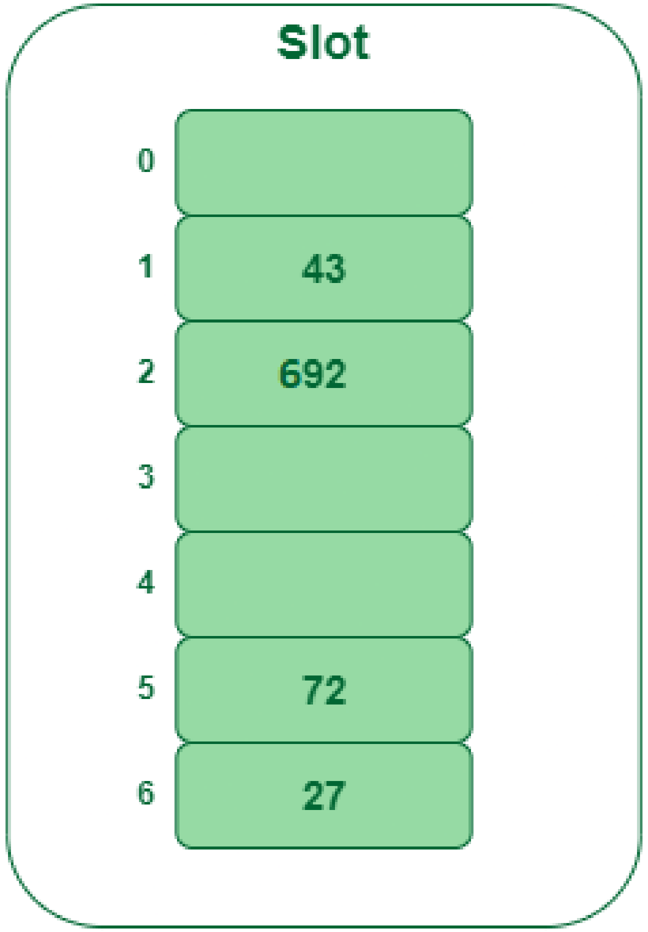
DOUBLE HASHING

- Insert 692
- 692 % 7 = 6, but location 6 is already being occupied and this is a collision
- So we need to resolve this collision using double hashing.



DOUBLE HASHING

- 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.



DOUBLE HASHING

Advantages

- No extra space is wasted
- No primary clustering
- No secondary clustering
- Double hashing is useful if an application requires a smaller hash table since it effectively finds a free slot.
- It produces a uniform distribution of records throughout a hash table.

Disadvantages

- O(n) time complexity

DOUBLE HASHING: EXAMPLE

- Suppose there are six students in the Data Structures class and their IDs are 115, 153, 586, 206, 985, and 111, respectively. We want to store each student's data in this order. Suppose HT is of the size 19.
- How will the data is stored in HT using Double Hashing