UNIT-V

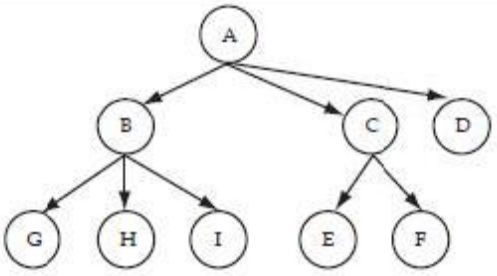
PART-A

1.What is a tree?

A tree is a set of one or more nodes T such that:

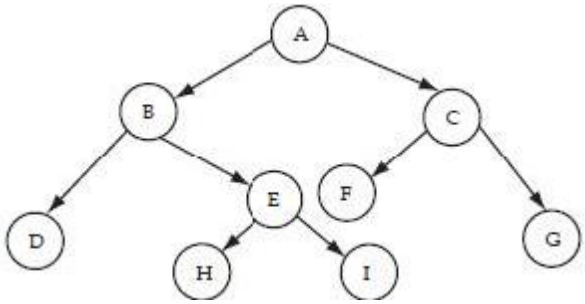
1. There is a specially designated node called root, and

2. Remaining nodes are partitioned into n >= 0 disjoint set of nodes T1, T2,…,Tn each of which is a tree.



2.What is a binary tree data structure?

A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called left subtree and right subtree. It is the most commonly used non-linear data structure



3.what are the properties of binary tree?

1. If a binary tree contains n nodes, then it contains exactly n – 1 edges;

2. A Binary tree of m levels has 2m– 1 nodes or less.

4.What are the types of binary tree representations?

A binary tree data structure is represented using two methods. Those methods are

1)Array Representation

2)Linked List Representation

5. what are the applications of trees?

Following are some important application of trees

Decision Tree – Machine learning Algorithm

Sorting

File Systems

Search Engines

Binary Expression Trees

Data Compression- Huffman coding

6.Name different types of the traversals.

There are three types of binary tree traversals.

1)In - Order Traversal

2)Pre - Order Traversal

3)Post - Order Traversal

7. What is binary search tree?

A binary search tree (BST) is a binary tree. It may be empty. If it is not empty, then all nodes follows the below mentioned properties −

- Every element has a unique key.
- The keys in a nonempty left subtree (right subtree) are smaller (larger) than the key in the root of subtree.
- The keys in a nonempty right subtree larger than the key in the root of subtree.

The left and right subtrees are also binary search trees.

8.what are the properties of binary search tree?

For any given node: All keys in its left subtree are less than the node's key.

All keys in its right subtree are greater than the node's key.

Performing an in-order traversal of a BST visits the keys in ascending sorted order.

9.Define Hash table.

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.

10.define hash function.

A hash function is a special mathematical function that takes a key and converts it into a specific index within the hash table. This function ensures that the same key always produces the same index. Good hash functions distribute keys evenly across the hash table to minimize collisions.

11. What are the types of hash functions

The types of hash functions are:

1. Division Method.
2. Mid Square Method.
3. Folding Method.
4. Multiplication Method.

12. What is collision?

When the hash function generates the same index for multiple keys, there will be a conflict (what value to be stored in that index). This is called a hash collision.

13. What are collision resolution techniques?

collisions in hash tables can be minimized using collision resolution techniques

1.Separate chaining

2.Open addressing

14. What are the Techniques used for open addressing?

The techniques used for open addressing are

–Linear Probing

–Quadratic Probing

–Double Hashing

15. What is double hashing?

Double hashing uses two hash functions,

– one to find the initial location to place the key and a

– second to determine the size of the jumps in the probe sequence.

PART-B

1. What is binary search tree? Explain its operations with example.

Definition: A binary search tree (BST) is a binary tree. It may be empty. If it is not empty, then all nodes follows the below mentioned properties −

- **Every element has a unique key.**
- **The keys in a nonempty left subtree (right subtree) are smaller (larger) than the key in the root of subtree.**
- **The keys in a nonempty right subtree larger than the key in the root of subtree.**
- **The left and right subtrees are also binary search trees.**

The basic operations that can be performed on binary search tree data structure, are following −

1. Search − search an element in a binary search tree.

2.Insert − insert an element into a binary search tree / create a tree.

3.Delete − Delete an element from a binary search tree.

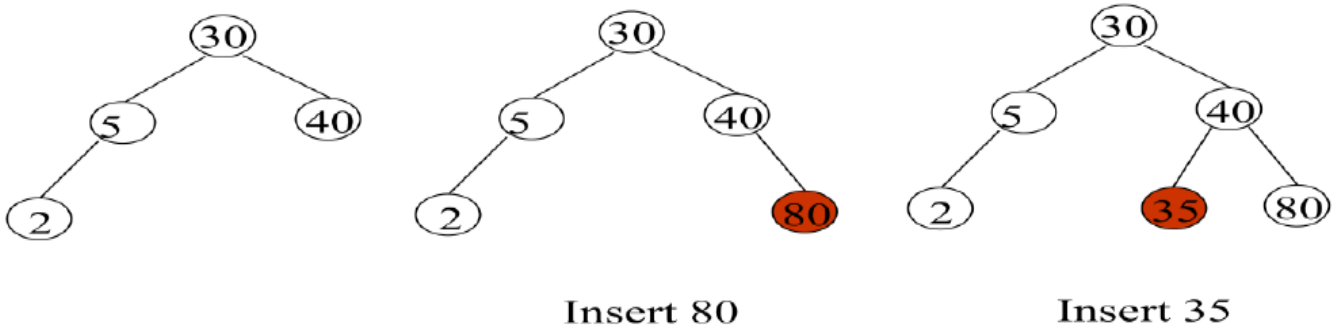
4. Traversal- visit and print every element in a binary search tree

Searching a Binary Search Tree

Let an element k is to search in binary search tree. Start search from root node of the search tree. If root is NULL, search tree contains no nodes and search unsuccessful. Otherwise, compare k with the key in the root. If k equals the root's key, terminate search, if k is less than key value, search element k in left subtree otherwise search element k in right subtree. The function search recursively searches the subtrees.

Inserting into a Binary Search Tree

The very first insertion creates the tree. Afterwards, whenever an element is to be inserted. First locate its proper location. Start search from root node then if data is less than key value, search empty location in left sub tree and insert the data. Otherwise search empty location in right sub tree and insert the data.



Deleting a node

Remove operation on binary search tree is more complicated, than insert and search. Basically, in can be divided into two stages:

- **search for a node to remove**
- **if the node is found, run remove algorithm.**

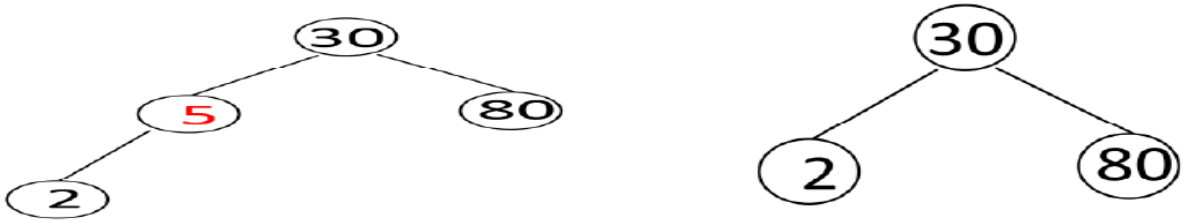
Remove algorithm in detail

Now, let's see more detailed description of a remove algorithm. First stage is identical to algorithm for lookup, except we should track the parent of the current node. Second part is more tricky. There are three cases, which are described below.

i.Node to be removed has no children. --This case is quite simple. Algorithm sets corresponding link of the parent to NULL and disposes the node.

Example. Remove -4 from a BST.

ii.Node to be removed has one child. In this case, node is cut from the tree and algorithm links single child (with it's subtree) directly to the parent of the removed node.



iii.Node to be removed has two children. --This is the most complex case. The deleted node can be replaced by either largest key in its left subtree or the smallest in its right subtree. Preferably which node has one child.

4. Traversal

Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.

There are three types of binary tree traversals.

In - Order Traversal

Pre - Order Traversal

Post - Order Traversal

We may perform In-order Traversal as it displays the nodes in ascending order.

In - Order Traversal

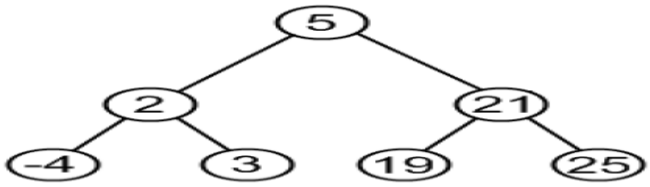
Until all nodes are traversed −

Step 1 − Recursively traverse left subtree.

Step 2 − Visit root node.

Step 3 − Recursively traverse right subtree.

Example:



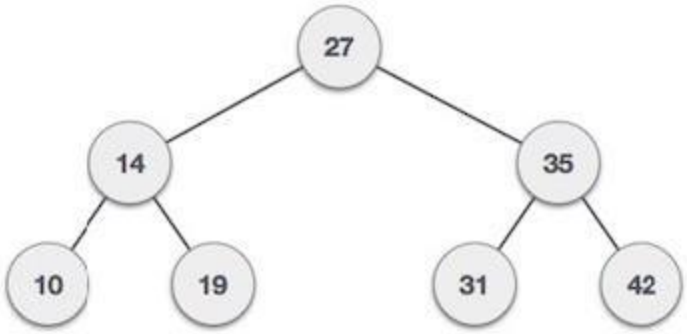
In-order: -4,2,3,5,19,21,25

2. what is binary search tree? Describe the process of inserting a node into a binary search tree.Write the algorithm for BST insertion.

Definition: A binary search tree (BST) is a binary tree. It may be empty. If it is not empty, then all nodes follows the below mentioned properties −

- **Every element has a unique key.**
- **The keys in a nonempty left subtree (right subtree) are smaller (larger) than the key in the root of subtree.**
- **The keys in a nonempty right subtree larger than the key in the root of subtree.**
- **The left and right subtrees are also binary search trees.**

Example of Binary search tree



Inserting into a Binary Search Tree

The very first insertion creates the tree. Afterwards, whenever an element is to be inserted. First locate its proper location. Start search from root node then if data is less than key value, search empty location in left sub tree and insert the data. Otherwise search empty location in right sub tree and insert the data.

In binary search tree, new node is always inserted as a leaf node. The insertion operation is performed as follows...

Step 1: Create a newNode with given value and set its left and right to NULL.

Step 2: Check whether tree is Empty.

Step 3: If the tree is Empty, then set set root to newNode.

Step 4: If the tree is Not Empty, then check whether value of newNode is smaller or larger than the node (here it is root node).

Step 5: If newNode is smaller than or equal to the node, then move to its left child. If newNode is larger than the node, then move to its right child.

Step 6: Repeat the above step until we reach a node (e.i., reach to NULL) where search terminates.

Step 7: After reaching a last node, then insert the newNode as left child if newNode is smaller or equal to that node else insert it as right child.

Algorithm

Create newnode

If root is NULL

then create root node

return

If root exists then

compare the data with node.data

while until insertion position is located

If data is greater than node.data

goto right subtree

else

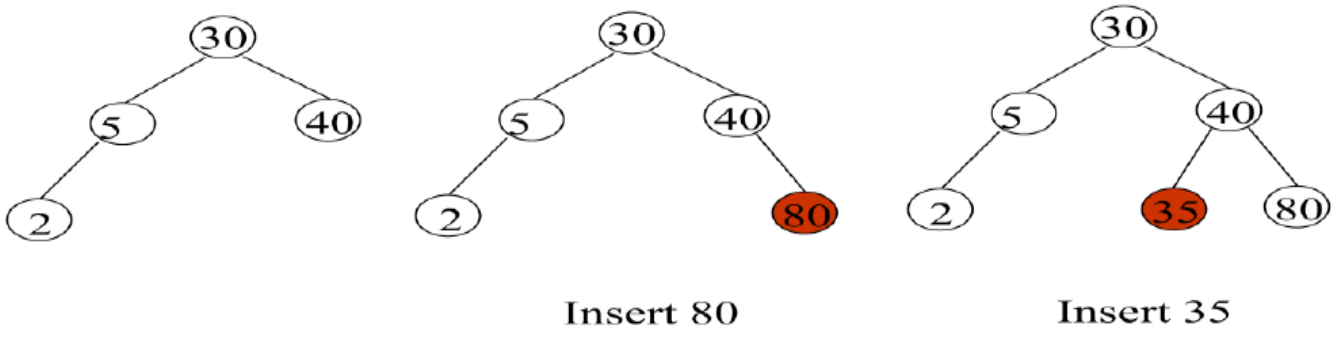
goto left subtree

endwhile

insert newnode

end If

Example:



3. what is binary search tree(BST)? Write a C function to implement BST insertion operation.

Definition: A binary search tree (BST) is a binary tree. It may be empty. If it is not empty, then all nodes follows the below mentioned properties −

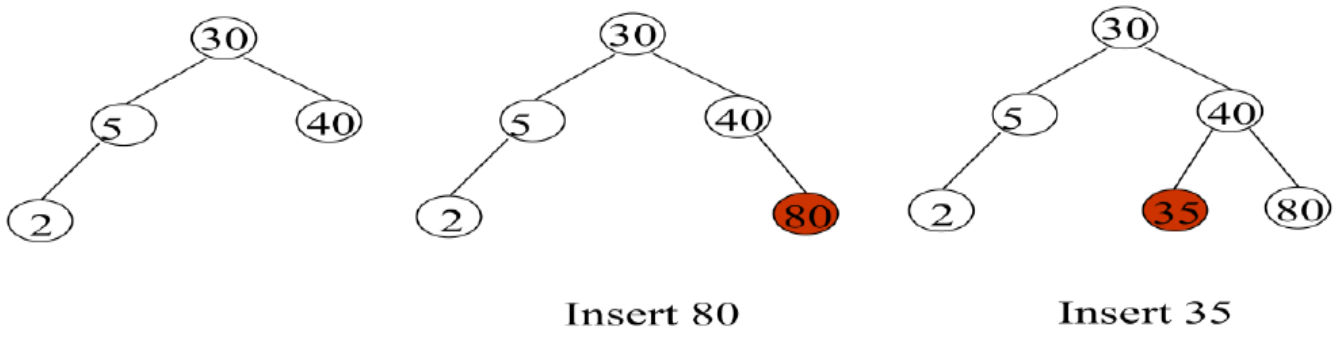
- **Every element has a unique key.**
- **The keys in a nonempty left subtree (right subtree) are smaller (larger) than the key in the root of subtree.**
- **The keys in a nonempty right subtree larger than the key in the root of subtree.**
- **The left and right subtrees are also binary search trees.**

Implementation of Insertion operation in BST

```
void insert(int data)
{
struct node *tempNode = (struct node*) malloc(sizeof(struct node));
struct node *current;
struct node *parent;
tempNode->data = data;
tempNode->leftChild = NULL;
tempNode->rightChild = NULL;
//if tree is empty, create root node
if(root == NULL) {
root = tempNode;
}else {
current = root;
parent = NULL;
while(1) {
parent = current;
if(data < parent->data) {
        current = current->leftChild;
```

```
            if(current == NULL) {
            parent->leftChild = tempNode;
            return;
             }
    }
else {
            current = current->rightChild;
            if(current == NULL) {
            parent->rightChild = tempNode;
            return;
            }
     }
 }
}
}
```

Example:



4. Describe the process of Deleting a node from a binary search tree with an example.

Deleting a node

Remove operation on binary search tree is more complicated, than insert and search. Basically, in can be divided into two stages:

- **search for a node to remove**
- **if the node is found, run remove algorithm.**

Remove algorithm in detail

Now, let's see more detailed description of a remove algorithm. First stage is identical to algorithm for lookup, except we should track the parent of the current node. Second part is more tricky. There are three cases, which are described below.

i.Node to be removed has no children. --This case is quite simple.

We use the following steps to delete a leaf node from BST...

Step 1: Find the node to be deleted using search operation

Step 2: Delete the node using free function (If it is a leaf) and terminate the function.

Example. Remove -4 from a BST.

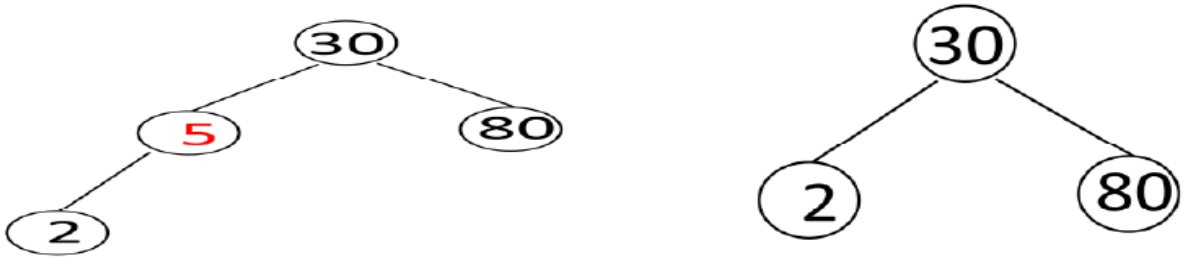
ii.Node to be removed has one child. In this case, node is cut from the tree and algorithm links single child (with it's subtree) directly to the parent of the removed node.

We use the following steps to delete a node with one child from BST...

Step 1: Find the node to be deleted using search operation

Step 2: If it has only one child, then create a link between its parent and child nodes.

Step 3: Delete the node using free function and terminate the function.



iii.Node to be removed has two children. --This is the most complex case. The deleted node can be replaced by either largest key in its left subtree or the smallest in its right subtree. Preferably which node has one child.

We use the following steps to delete a node with two children from BST...

Step 1: Find the node to be deleted using search operation

Step 2: If it has two children, then find the largest node in its left subtree (OR) the smallest node in its right subtree.

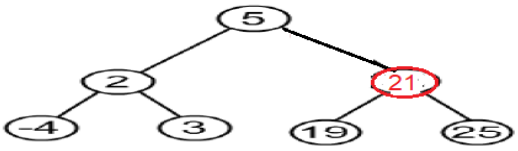
Step 3: Swap both deleting node and node which found in above step.

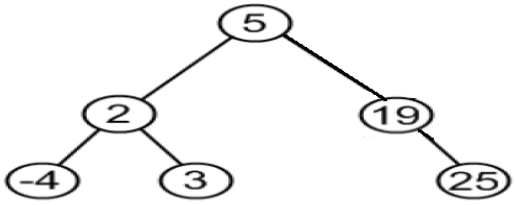
Step 4: Then, check whether deleting node came to case 1 or case 2 else goto steps 2

Step 5: If it comes to case 1, then delete using case 1 logic.

Step 6: If it comes to case 2, then delete using case 2 logic.

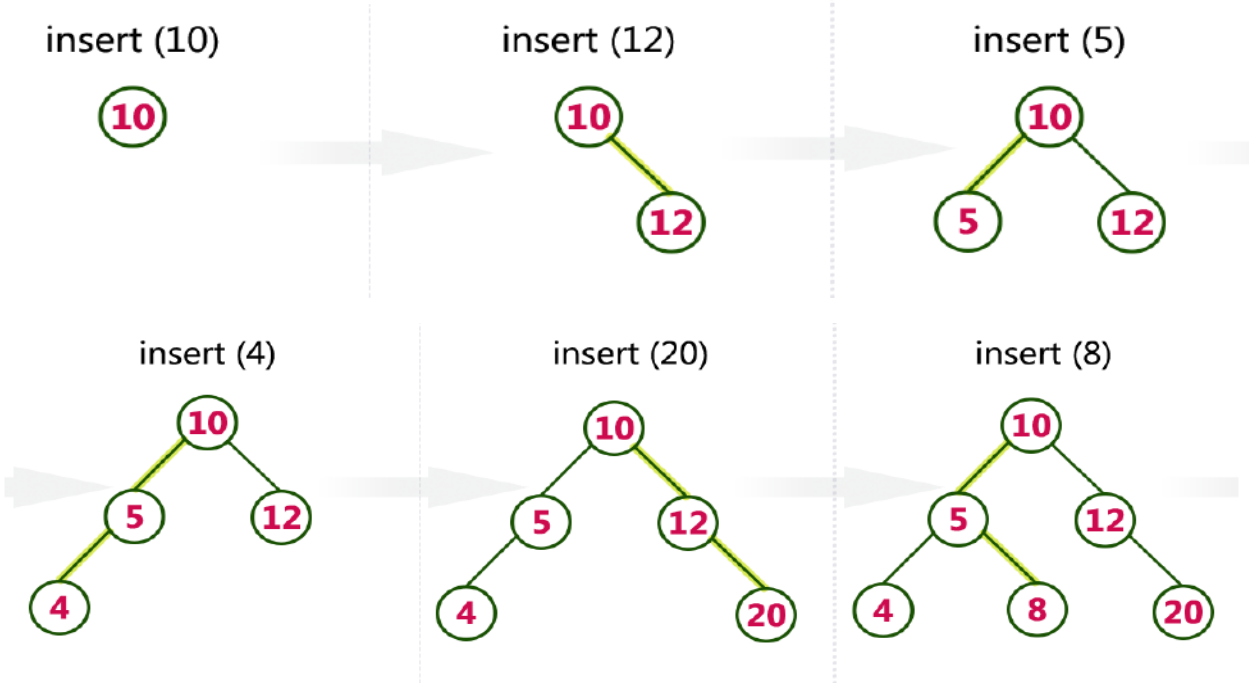
Step 7: Repeat the same process until node is deleted from the tree.

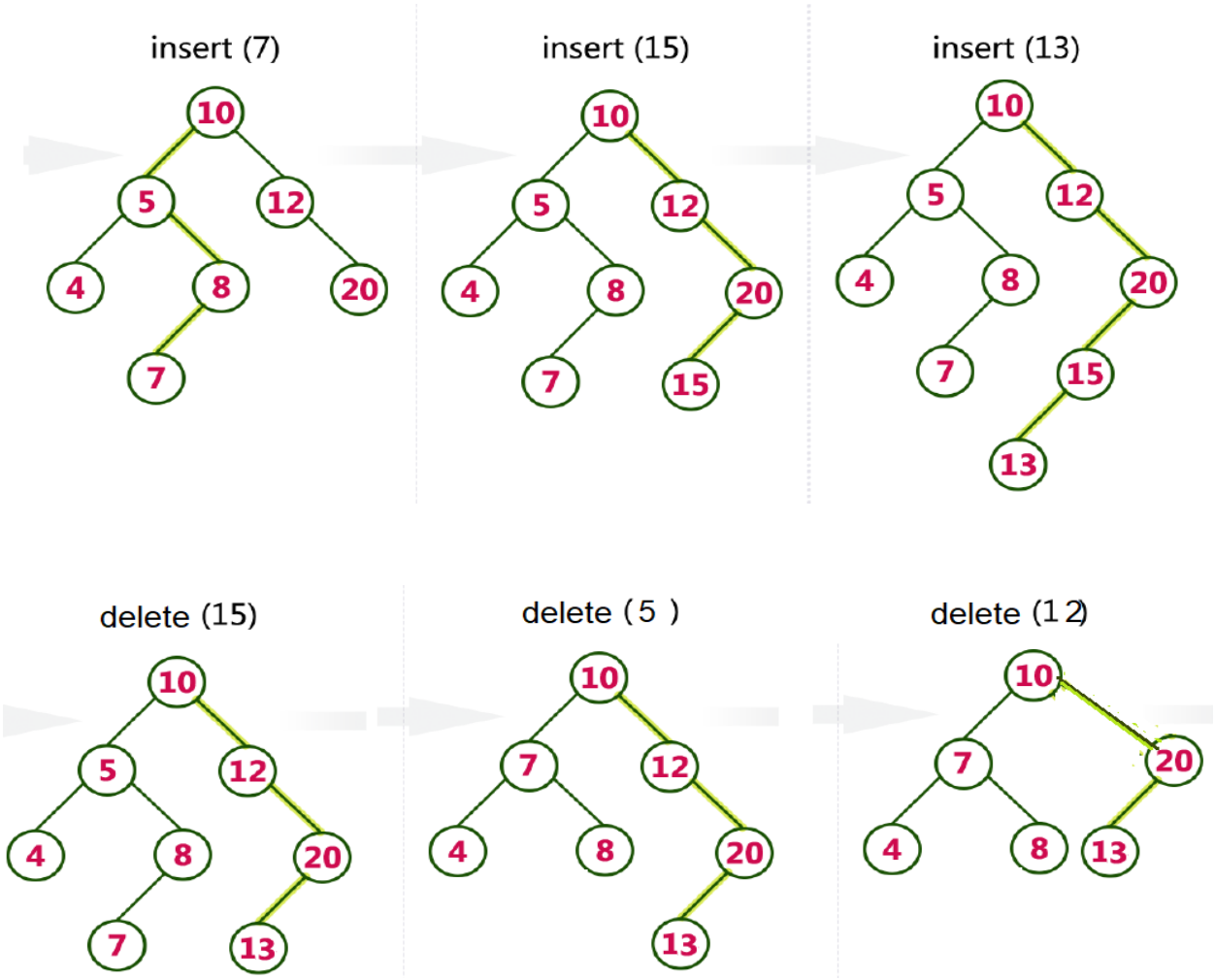




5. Construct a Binary Search Tree by inserting the following sequence of numbers...

10,12,5,4,20,8,7,15 and 13 Apply delete operation on the following key elements 13,20 and 5.



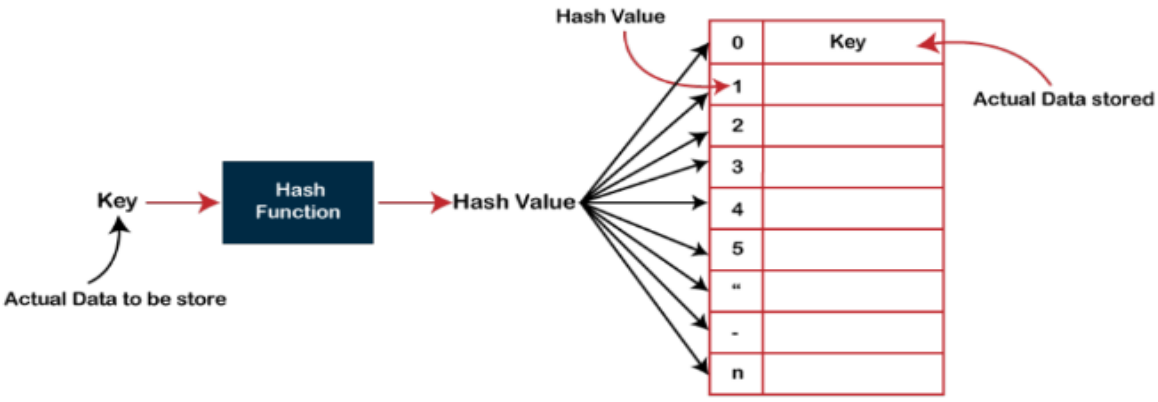


6. Explain the basic implementation of hash tables and the operations supported by hash tables, such as insertion, deletion and search.

In Hashing technique, the hash table and hash function are used. Using the hash function, we can calculate the address at which the value can be stored.

The main idea behind the hashing is to create the (key/value) pairs. If the key is given, then the algorithm computes the index at which the value would be stored. It can be written as:

Index = hash(key) % array-size.



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.

The hash table can be implemented with the help of an array, each location of hash table is referred as bucket. Hash table uses a special function known as a hash function that maps a given value with a key to a location in a hash table.

Hash function takes the key as an input and returns a small integer value as an output.

- The small integer value is called as a hash value.
- Hash value of the key is then used as an index for storing it into the hash table.

Operations of a hash table are

Search − Searches an element in a hash table.

Insert − inserts an element in a hash table.

Delete − Deletes an element from a hash table.

Assume a table with 8 slots:

Hash key = key % table size

36 % 8 = 4

18 % 8 = 2

72 % 8 = 0

[0]	72
[1]	
[2]	18
[3]	
[4]	36
[5]	
[6]	
[7]	

1. Search (Finding a Value by Key):

- Calculate the hash value of the key using the hash function.
- Use the hash value as the index to locate the potential value in the hash table.
- If the location contains the key, return the corresponding value.
- If the location is empty or contains a different key (collision), use a collision resolution technique to find the correct value.

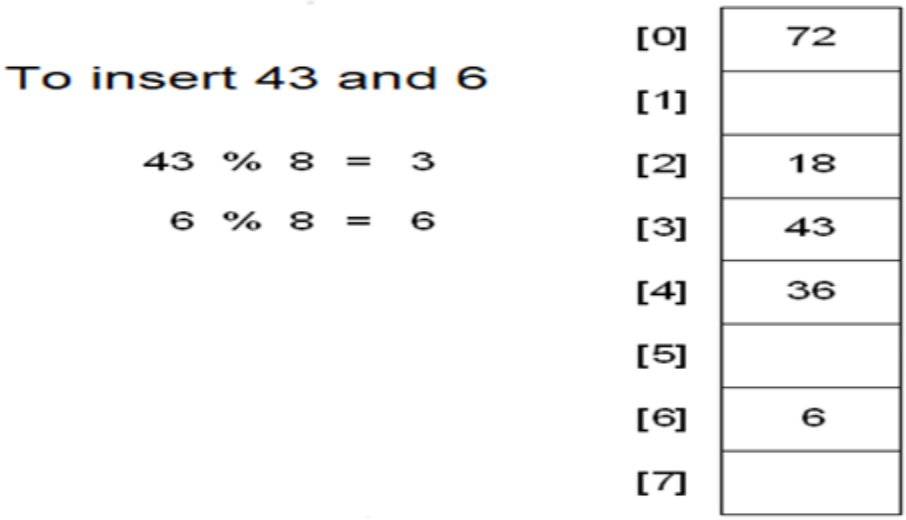
Example: To search a key value 36 hash value is computed, 36 % 8 = 4;

4 is used as index to locate the value in hash table.

2.Insertion (Adding a Key-Value Pair):

- Calculate the hash value of the key using the hash function.
- Use the hash value as the index to determine the location in the hash table where the value will be stored.
- If the location is empty, store the key-value pair there.
- If the location is occupied (collision), use a collision resolution technique (e.g., separate chaining, open addressing).

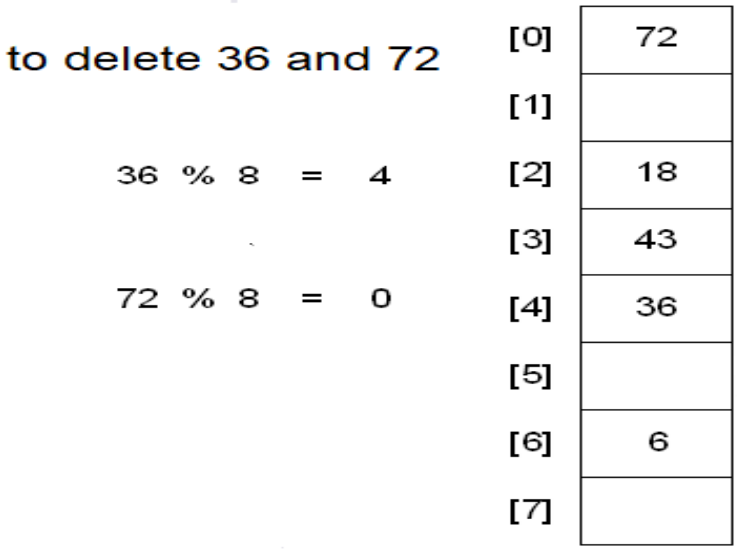
Example:



3. Deletion (Removing a Key-Value Pair):

- Calculate the hash value of the key using the hash function.
- Use the hash value as the index to locate the key-value pair in the hash table.
- If the location contains the key, remove the key-value pair.
- If the location is empty or contains a different key (collision), use a collision resolution technique to find the correct key-value pair to delete.

Example



7. What is hash function? Explain the different hash functions with examples.

Types of Hash Functions

The primary types of hash functions are:

Division Method.

Mid Square Method.

Folding Method.

Multiplication Method.

Division Method

The easiest and quickest way to create a hash value is through division. The k-value is divided by M in this hash function, and the result is used.

Formula:

h(K) = k mod M

(where k = key value and M = the size of the hash table)

Advantages:

This method is effective for all values of M.

The division strategy only requires one operation, thus it is quite quick.

Disadvantages:

Since the hash table maps consecutive keys to successive hash values, this could result in poor performance.

There are times when exercising extra caution while selecting M's value is necessary.

Example of Division Method

k = 1987

M = 13

h(1987) = 1987 mod 13

h(1987) = 4

Mid Square Method

The following steps are required to calculate this hash method:

k*k, or square the value of k

Using the middle r digits, calculate the hash value.

Formula:

h(K) = h(k x k)

(where k = key value)

Advantages:

This technique works well because most or all of the digits in the key value affect the result.

All of the necessary digits participate in a process that results in the middle digits of the squared result.

The result is not dominated by the top or bottom digits of the initial key value.

Disadvantages:

The size of the key is one of the limitations of this system; if the key is large, its square will contain twice as many digits.

Probability of collisions occurring repeatedly.

Example of Mid Square Method

k = 60

Therefore,k = k x k

k = 60 x 60

k = 3600

Thus, h(60) = 60

Folding Method

The process involves two steps:

except for the last component, which may have fewer digits than the other parts, the key-value k should be divided into a predetermined number of pieces, such as k1, k2, k3,..., kn, each having the same amount of digits.

Add each element individually. The hash value is calculated without taking into account the final carry, if any.

Formula:

k = k1, k2, k3, k4, ….., kn

s = k1+ k2 + k3 + k4 +….+ kn

h(K)= s

(Where, s = addition of the parts of key k)

Advantages:

Creates a simple hash value by precisely splitting the key value into equal-sized segments.

Without regard to distribution in a hash table.

Disadvantages:

When there are too many collisions, efficiency can occasionally suffer.

Example of Folding Method

k = 12345

k1 = 67; k2 = 89; k3 = 12

Therefore,s = k1 + k2 + k3

s = 67 + 89 + 12

s = 168

Multiplication Method

Determine a constant value. A, where (0, A, 1)

Add A to the key value and multiply.

Consider kA's fractional portion.

Multiply the outcome of the preceding step by M, the hash table's size.

Formula:

h(K) = floor (M (kA mod 1))

(Where, M = size of the hash table, k = key value and A = constant value)

Advantages:

Any number between 0 and 1 can be applied to it, however, some values seem to yield better outcomes than others.

Disadvantages:

The multiplication method is often appropriate when the table size is a power of two since multiplication hashing makes it possible to quickly compute the index by key.

Example of Multiplication Method

k = 5678

A = 0.6829

M = 200

Now, calculating the new value of h(5678):

h(5678) = floor[200(5678 x 0.6829 mod 1)]

h(5678) = floor[200(3881.5702 mod 1)]

h(5678) = floor[200(0.5702)]

h(5678) = floor[114.04]

h(5678) = 114

So, with the updated values, h(5678) is 114

If the key is 123, the table size is 10, and A is 0.618, then you calculate (123 * 0.618) % 1, get the fractional part, multiply by 10, and take the floor value to get the index.

8.What is a collision? What are the different collision resolution techniques? Explain Separate chaining with examples.

Collision Resolution

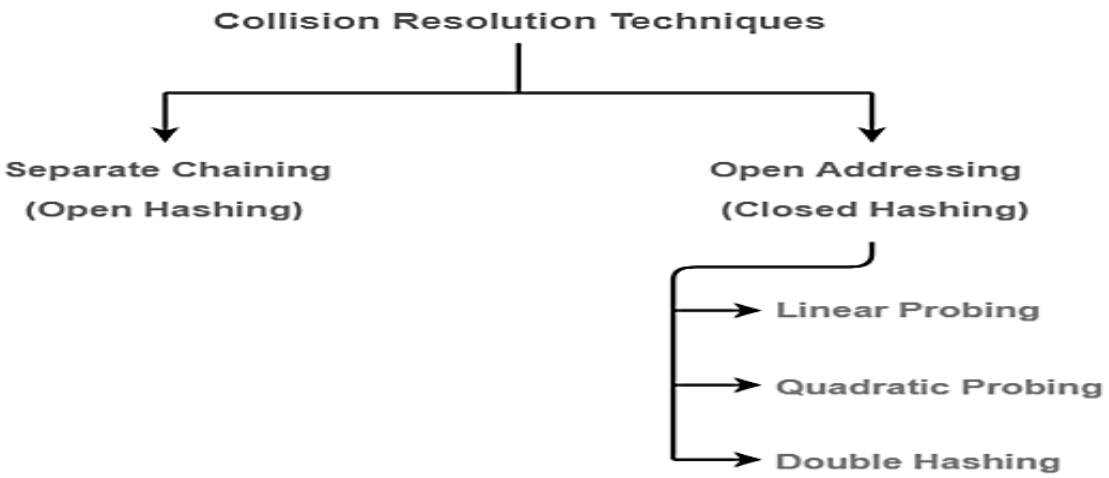
When two keys or hash values compete with a single hash table slot, then Collision occur. To resolve collision, we use collision resolution techniques. Collisions can be reduced with a selection of a good hash function.

Collision Resolution Techniques

There are two types of collision resolution techniques.

Separate chaining (open hashing)

Open addressing (closed hashing)



Separate Chaining

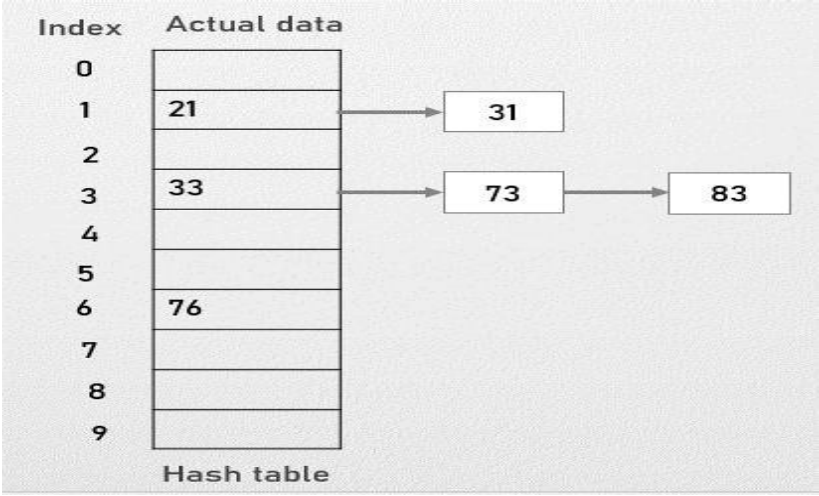
To handle the collision,

–This technique creates a linked list to the slot for which collision occurs.

–The new key is then inserted in the linked list.

–These linked lists to the slots appear like chains.

–That is why, this technique is called as separate chaining.



For Searching

In worst case, all the keys might map to the same bucket of the hash table.

In such a case, all the keys will be present in a single linked list.

Sequential search will have to be performed on the linked list to perform the search.

Worst case complexity for searching is O(n).

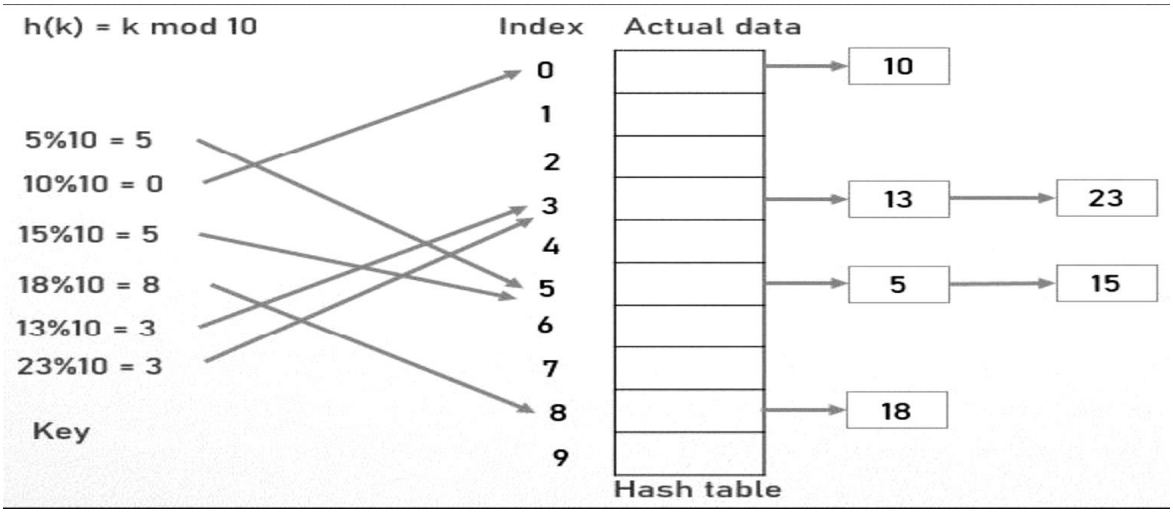
For Deletion

In worst case, the key might have to be searched first and then deleted.

In worst case, time taken for searching is O(n).

Worst case complexity for deletion is O(n).

Example-Separate Chaining



Advantages of separate chaining

- It is easy to implement.
- The hash table never fills full, so we can add more elements to the chain.
- It is mostly used when it is unknown how many and how frequently keys may be inserted or
- deleted.
- It is less sensitive to the function of the hashing.

Disadvantages of separate chaining

- The cache performance of chaining is not good.
- The memory wastage is too much in this method.
- It requires more space for element links.
- If the chain becomes long, then search time can become O(n) in the worst case.

9.What is a collision? What are the different collision resolution techniques? Explain open addressing with examples.

Collision:

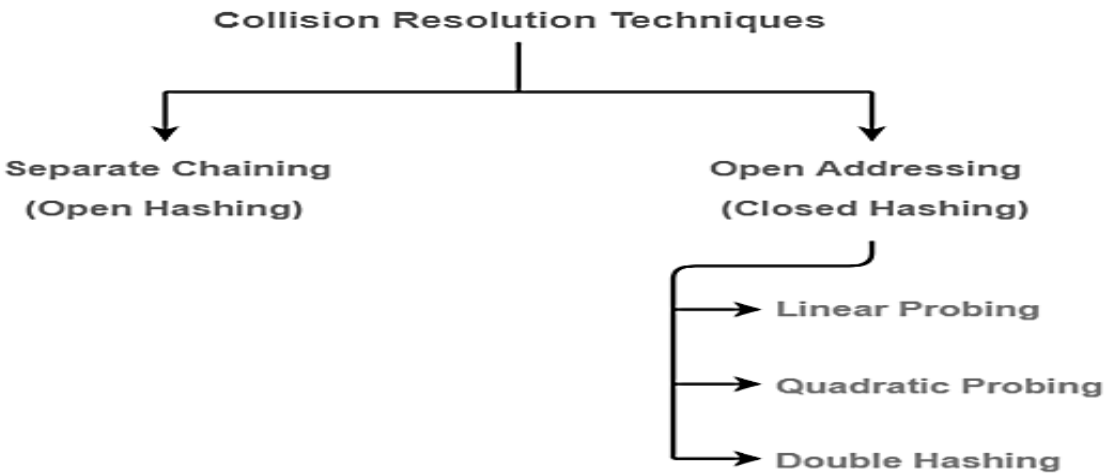
When two keys or hash values compete with a single hash table slot, then Collision occur. To resolve collision, we use collision resolution techniques. Collisions can be reduced with a selection of a good hash function.

Collision Resolution Techniques

There are two types of collision resolution techniques.

Separate chaining (open hashing)

Open addressing (closed hashing)



Open Addressing

•In open addressing,

–Unlike separate chaining, all the keys are stored inside the hash table.

–No key is stored outside the hash table.

•Techniques used for open addressing are-

–Linear Probing

–Quadratic Probing

–Double Hashing

Operations in Open Addressing

•Insert Operation:

–Hash function is used to compute the hash value for a key to be inserted.

–Hash value is then used as an index to store the key in the hash table.

•In case of collision,

–Probing is performed until an empty bucket is found.

–Once an empty bucket is found, the key is inserted.

–Probing is performed in accordance with the technique used for open addressing.

Search Operation:

•To search any particular key,

–Its hash value is obtained using the hash function used.

–Using the hash value, that bucket of the hash table is checked.

–If the required key is found, the key is searched.

–Otherwise, the subsequent buckets are checked until the required key or an empty bucket is found.

–The empty bucket indicates that the key is not present in the hash table.

Delete Operation:

–The key is first searched and then deleted.

–After deleting the key, that particular bucket is marked as “deleted”.

1. Linear Probing

In linear probing,

–When collision occurs, we linearly probe for the next bucket.

–We keep probing until an empty bucket is found.

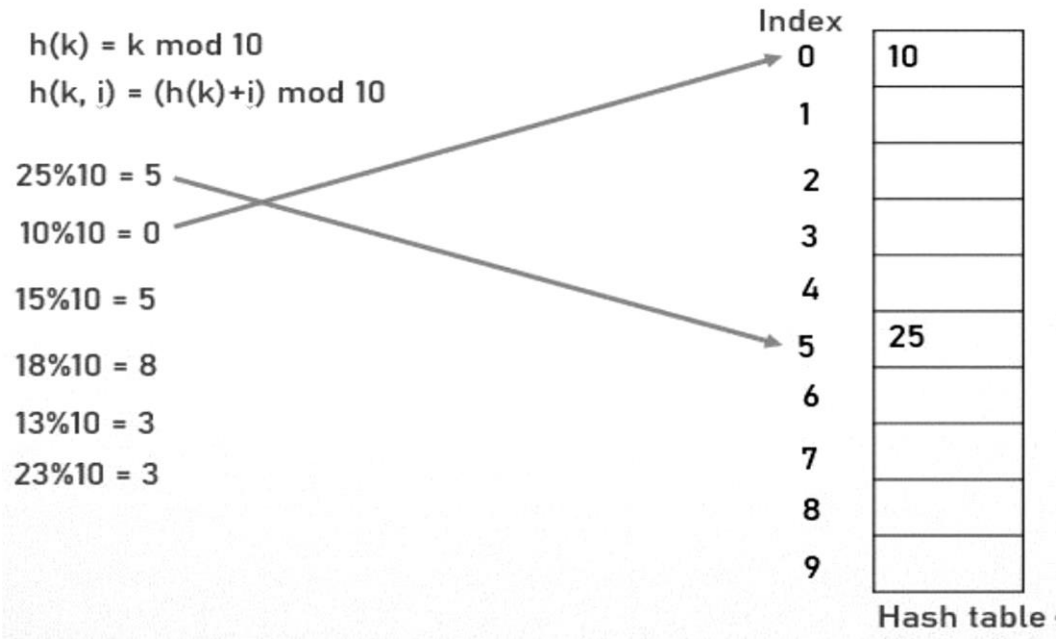
–Let hash(x) be the slot index computed using a hash function and n be the table size

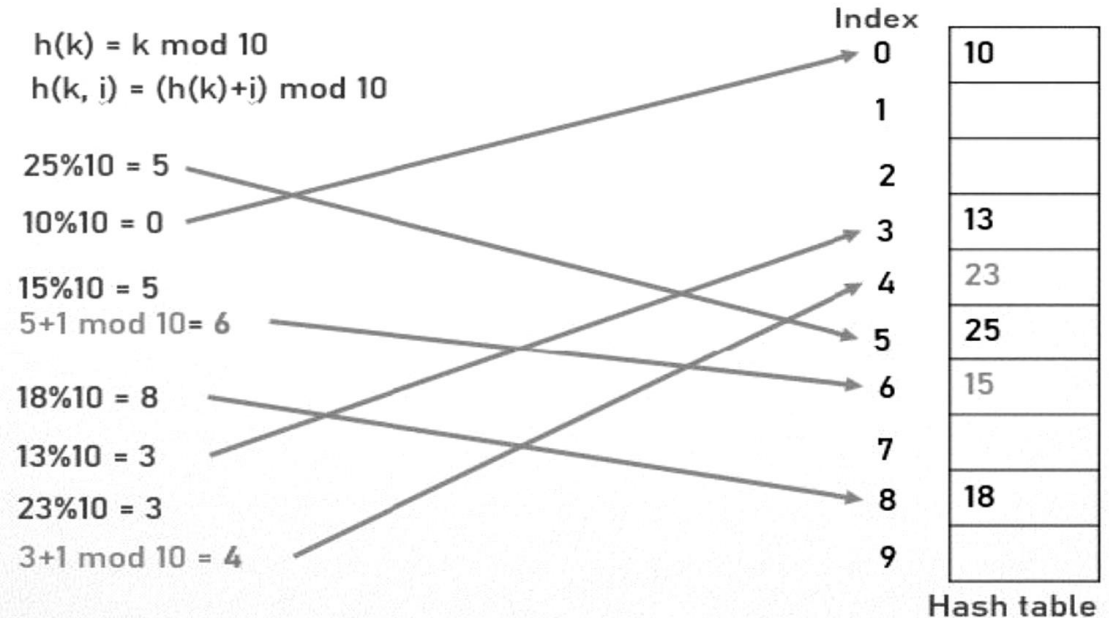
–If slot hash(x) % n is full, then we try (hash(x) + 1) % n

–If (hash(x) + 1) % n is also full, then we try (hash(x) + 2) % n

–If (hash(x) + 2) % n is also full, then we try (hash(x) + 3) % n so on…

Example: Linear Probing





•**Advantage-**

–It is easy to compute.

•**Disadvantage-**

–The main problem with linear probing is clustering.

–Many consecutive elements form groups.

–Then, it takes time to search an element or to find an empty bucket.

2. Quadratic Probing

In quadratic probing,

–When collision occurs, we probe for i^2'th bucket in ithiteration.

–We keep probing until an empty bucket is found.

–In quadratic probing the sequence is that $H+1^2$, $H+2^2$, $H+3^2$,....$H+K^2$

–The hash function for quadratic probing is

$h_i(X)$ = (Hash(X) + $(i)^2$) % Table Size for i = 0, 1, 2, 3,...etc.

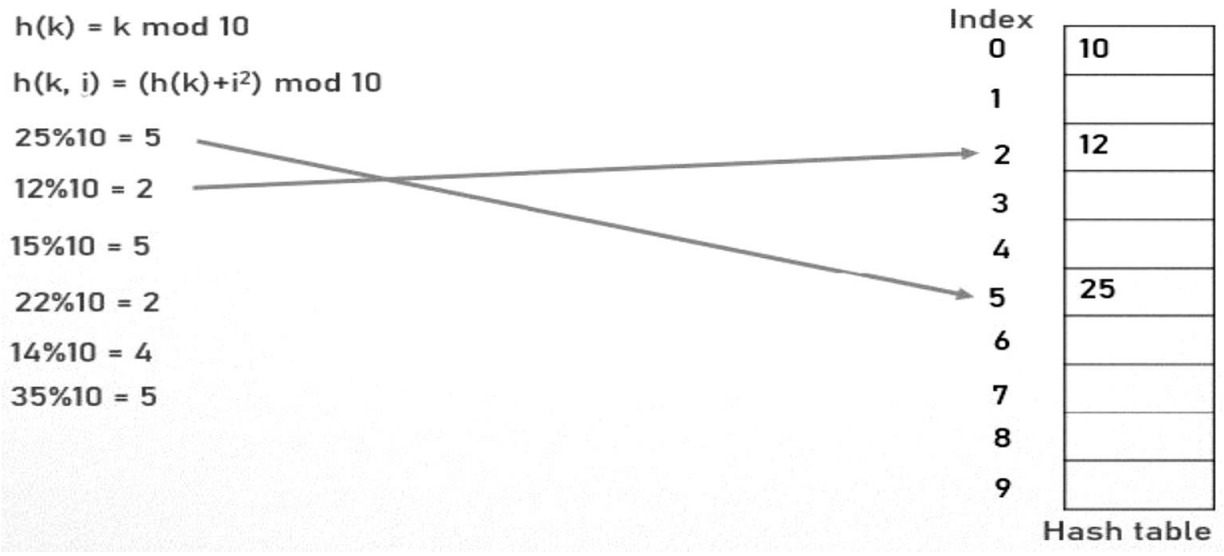
–If the slot hash(x) % S is full, then we try (hash(x) + 1*1) % S.

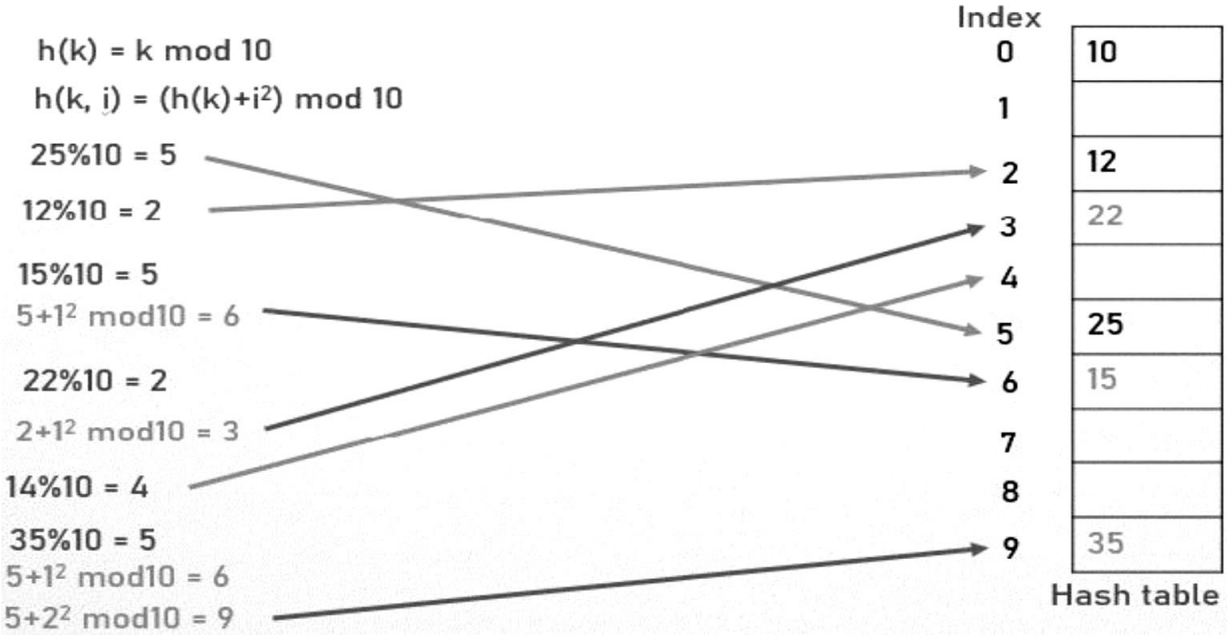
–If (hash(x) + 1*1) % S is also full, then we try (hash(x) + 2*2) % S.

–If (hash(x) + 2*2) % S is also full, then we try (hash(x) + 3*3) % S.

–This process continue until an empty slot is found.

Example : Quadratic probing





Advantage:

– Primary clustering problem resolved

Disadvantage:

– Secondary clustering

– No guarantee for finding slots

3.Double Hashing

Double hashing uses two hash functions,

– one to find the initial location to place the key and a

– second to determine the size of the jumps in the probe sequence.

–The ith probe is

h(k, i) = (hash1(key) + i * hash2(key)) mod TABLE_SIZE

Here hash1() and hash2() are hash functions

First hash function is:

hash1(key) = key mod TABLE_SIZE

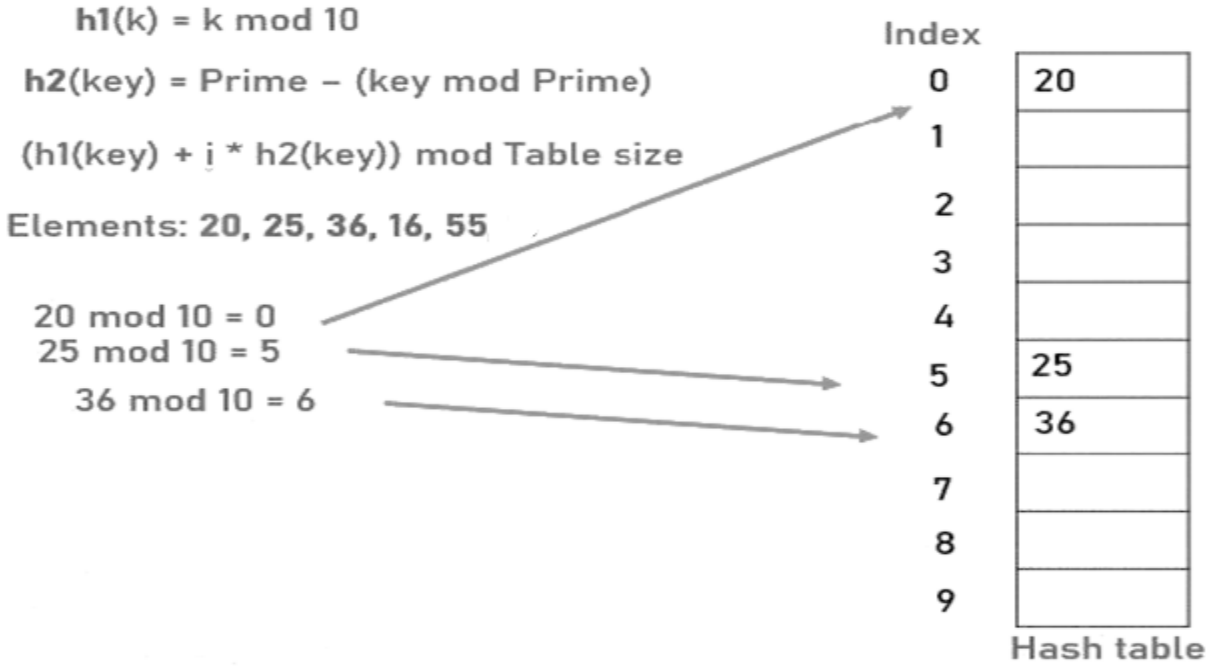
Second hash function is :

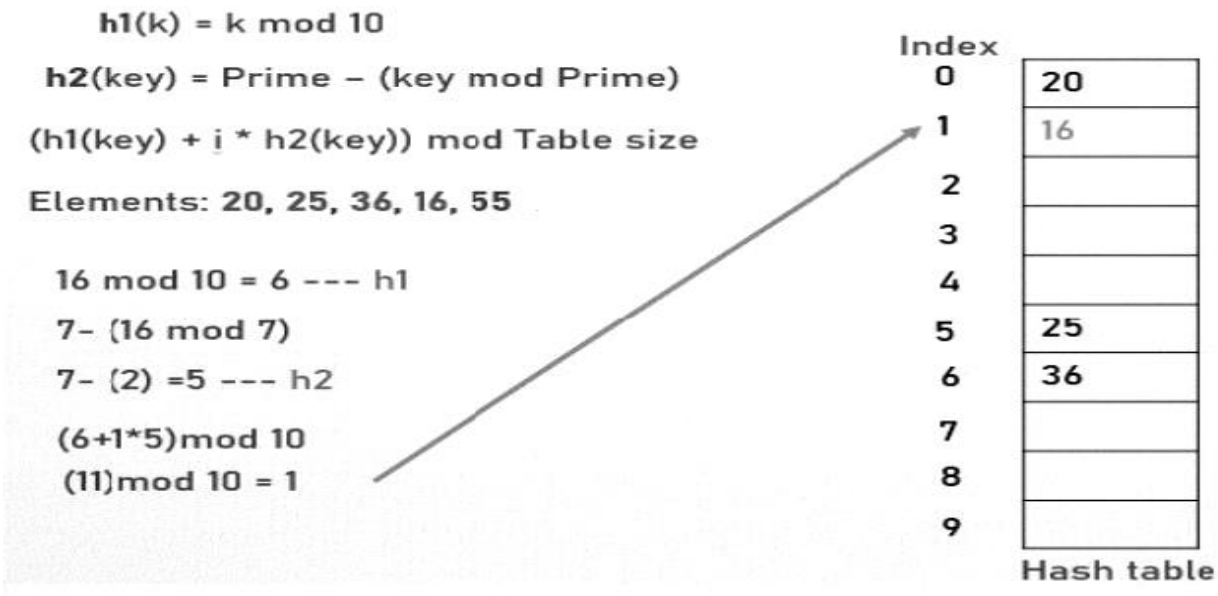
hash2(key) = PRIME – (key mod PRIME)

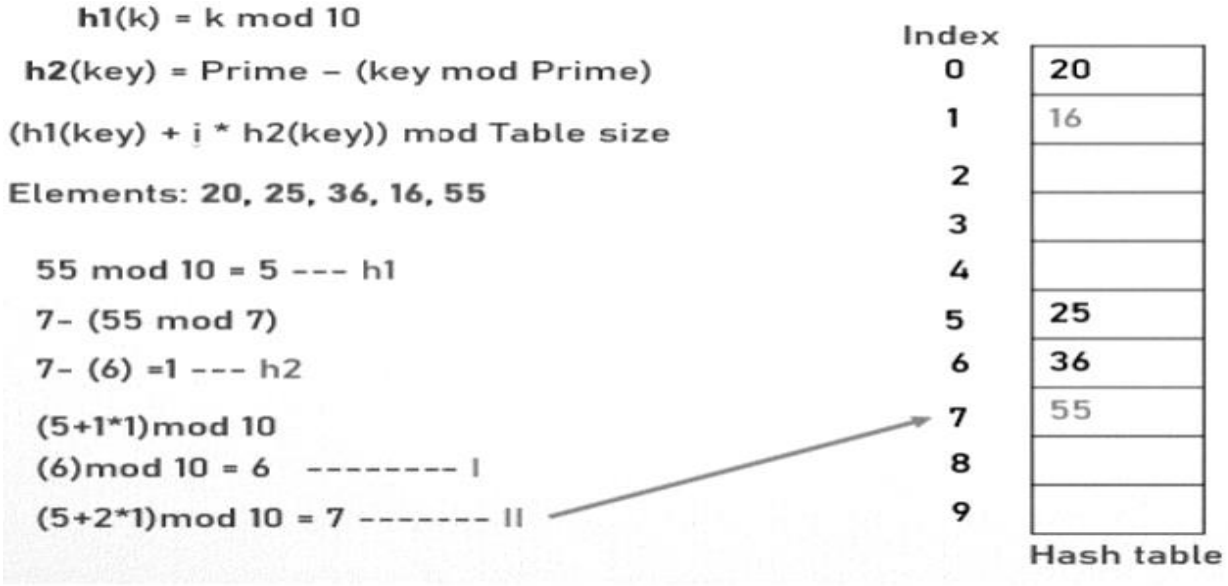
Where PRIME is a prime smaller than the TABLE_SIZE

Example: Double hashing

Elements: 20, 25, 36, 16, 55, 17 table size= 10 prime number = 7







Double hashing highlights

– Computational cost is higher.

– No primary clustering.

– No secondary clustering.

– Double hashing can find the next free slot faster than the linear probing approach.

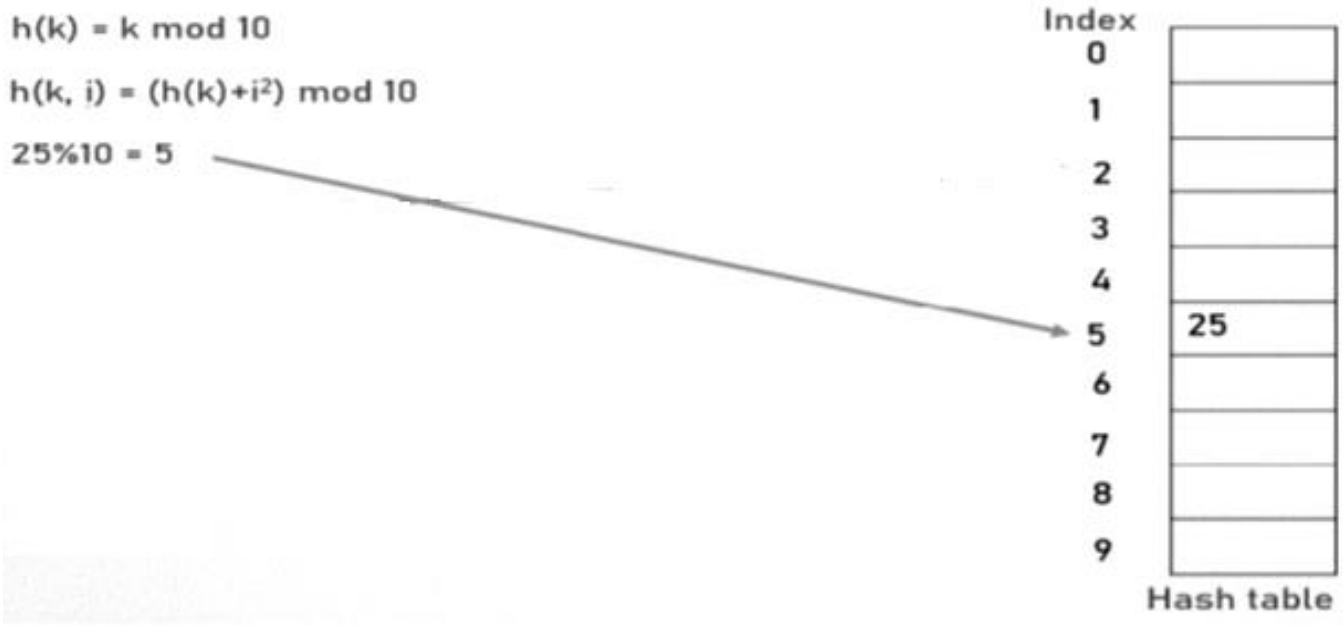
– Double hashing is used for uniform distribution of records throughout a hash table.

– Double hashing is useful if an application requires a smaller hash table.

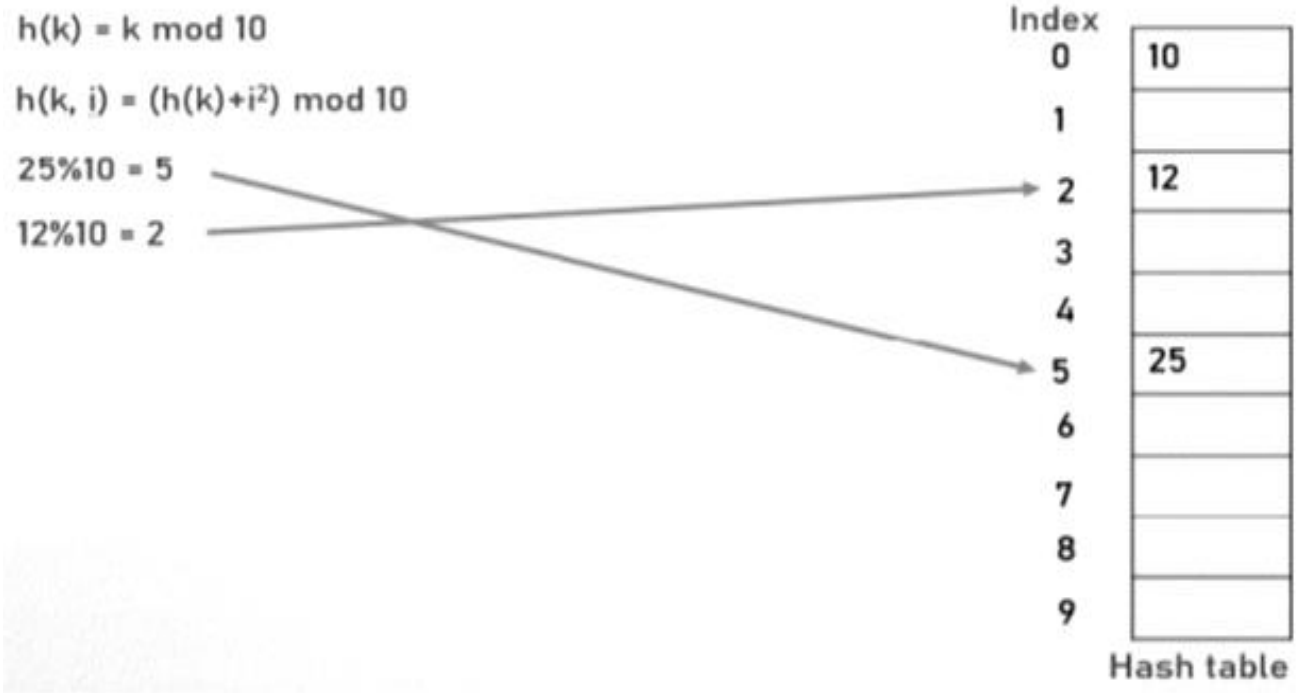
10.using the hash function key mod 10 insert the following sequence of keys in the hash table

25,12,15,22,14 and 35 use quadratic probing technique for collision resolution.

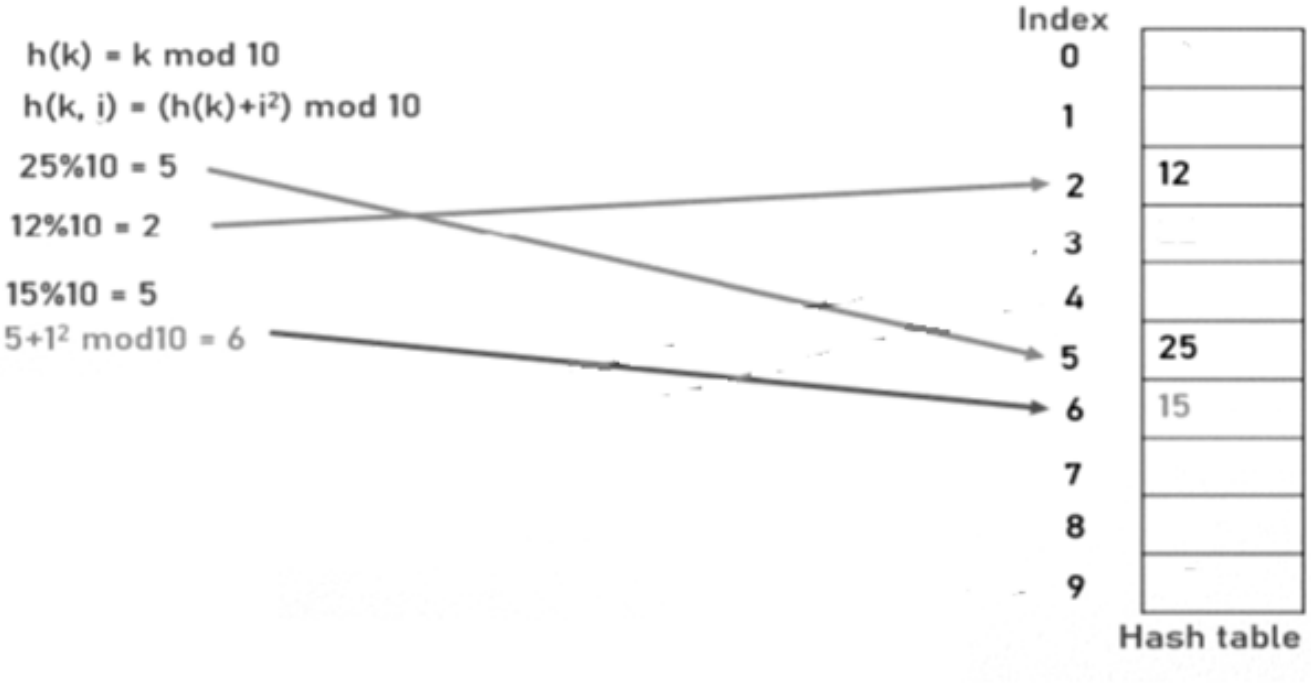
Step-1



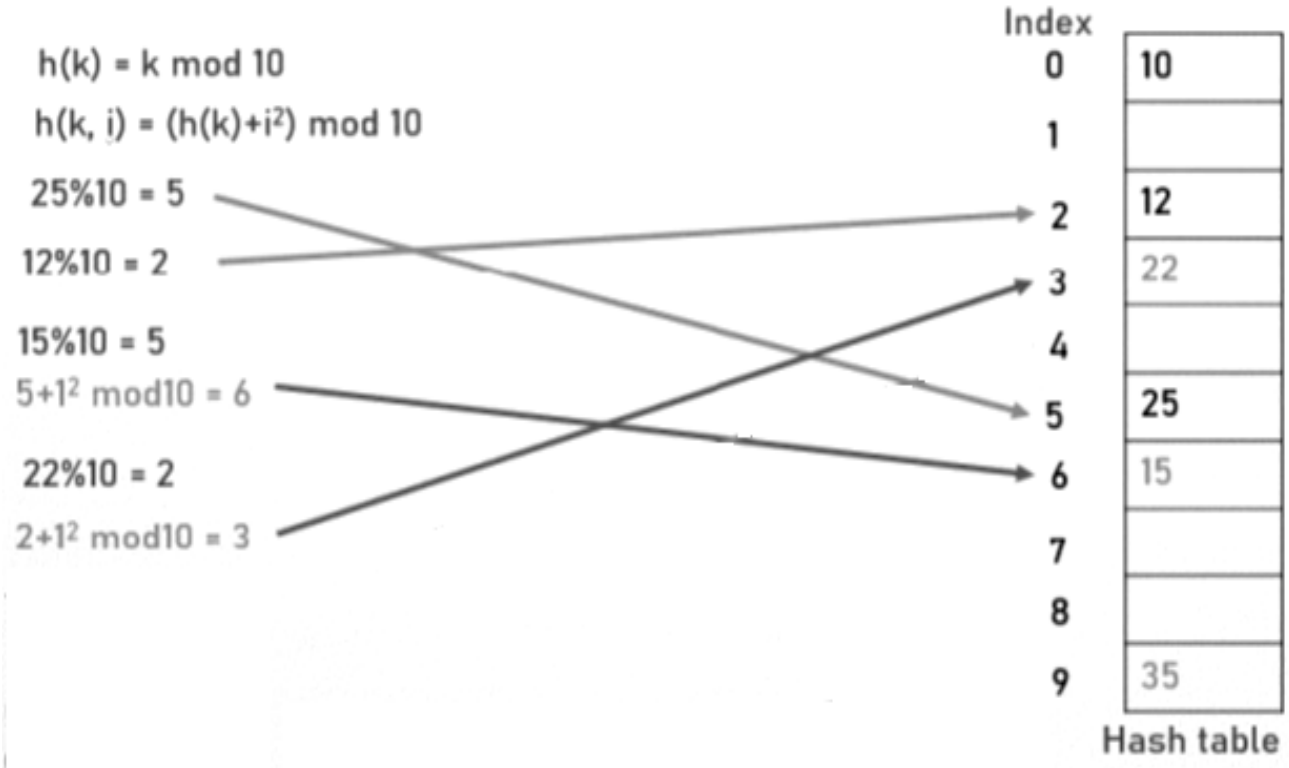
Step-2



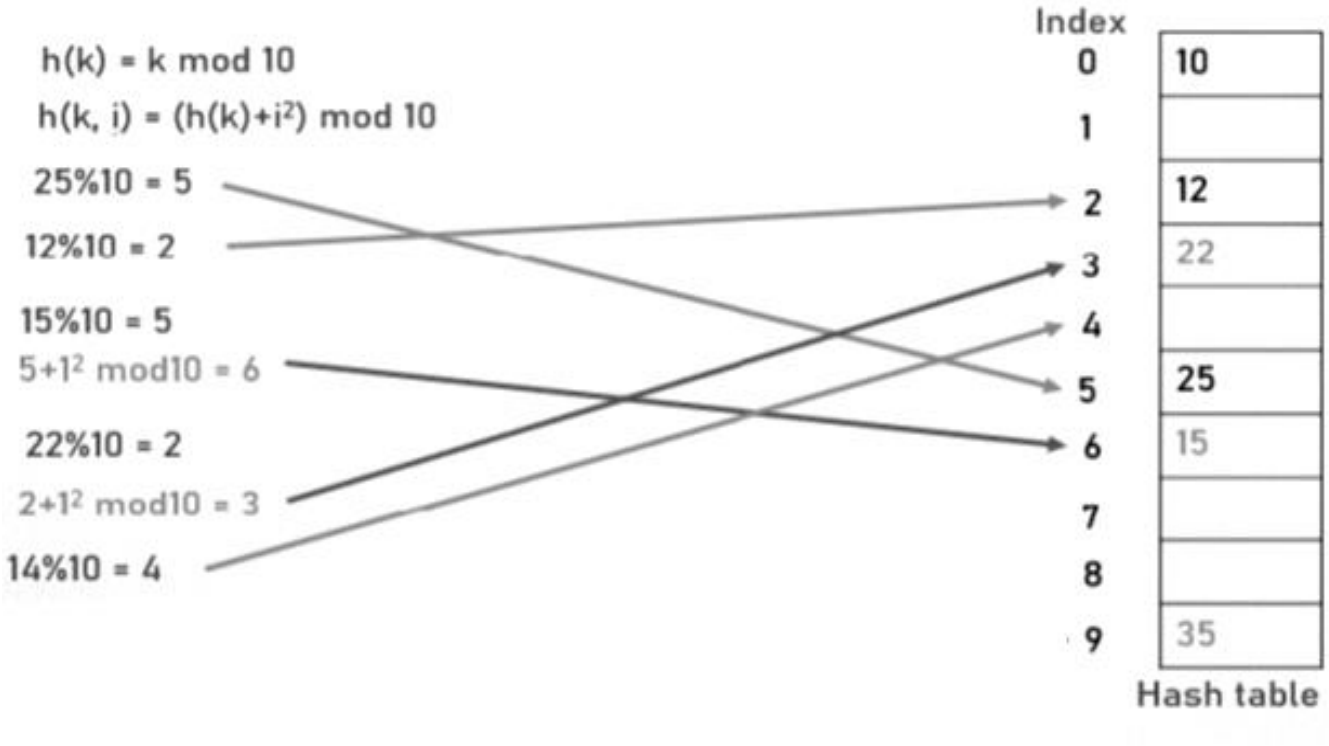
Step-3


h(k) = k mod 10
h(k, i) = (h(k)+i²) mod 10
25%10 = 5
12%10 = 2
15%10 = 5
5+1² mod10 = 6
Index
0
1
2
3
4
5
6
7
8
9
12
25
15
Hash table

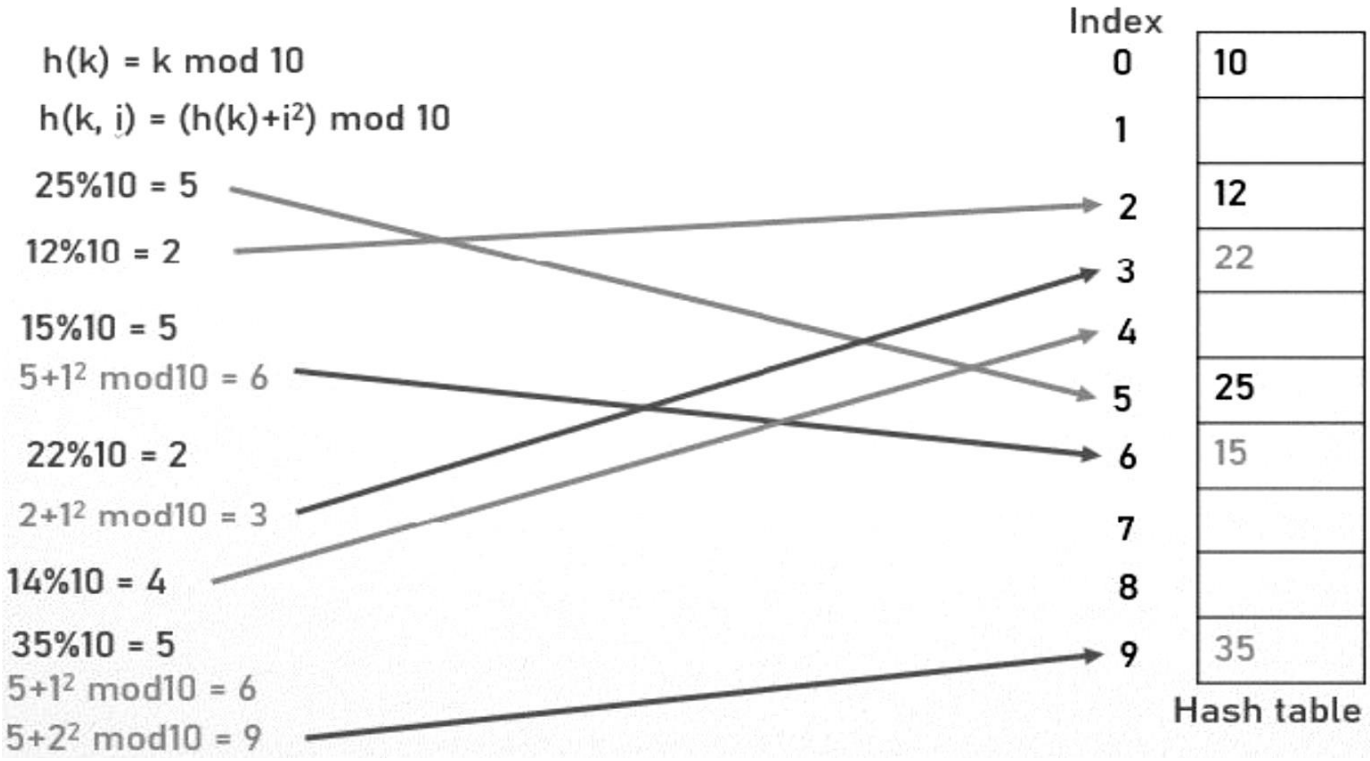
Step-4


h(k) = k mod 10
h(k, i) = (h(k)+i²) mod 10
25%10 = 5
12%10 = 2
15%10 = 5
5+1² mod10 = 6
22%10 = 2
2+1² mod10 = 3
Index
0
1
2
3
4
5
6
7
8
9
10
12
22
25
15
35
Hash table

Step-5


h(k) = k mod 10
h(k, i) = (h(k)+i²) mod 10
25%10 = 5
12%10 = 2
15%10 = 5
5+1² mod10 = 6
22%10 = 2
2+1² mod10 = 3
14%10 = 4
Index
0
1
2
3
4
5
6
7
8
9
10
12
22
25
15
35
Hash table

Step-6


h(k) = k mod 10
h(k, i) = (h(k)+i²) mod 10
25%10 = 5
12%10 = 2
15%10 = 5
5+1² mod10 = 6
22%10 = 2
2+1² mod10 = 3
14%10 = 4
35%10 = 5
5+1² mod10 = 6
5+2² mod10 = 9
Index
0
1
2
3
4
5
6
7
8
9
10
12
22
25
15
35
Hash table