**What is Hashing?**

A hash is a value that has a fixed length, and it is generated using a mathematical formula. Hash values are used in data compression, cryptology, etc. In data indexing, hash values are used because they have fixed length size regardless of the values that were used to generate them. It makes hash values to occupy minimal space compared to other values of varying lengths.

A hash function employs a mathematical algorithm to convert the key into a hash. A collision occurs when a hash function produces the same hash value for more than one key.

In this Algorithm tutorial, you will learn:

* [What is Hashing?](https://www.guru99.com/hash-table-data-structure.html#1)
* [What is a Hash Table?](https://www.guru99.com/hash-table-data-structure.html#2)
* [Hash functions](https://www.guru99.com/hash-table-data-structure.html#3)
* [Qualities of a good hash function](https://www.guru99.com/hash-table-data-structure.html#4)
* [Collision](https://www.guru99.com/hash-table-data-structure.html#5)
* [Hash table operations](https://www.guru99.com/hash-table-data-structure.html#6)
* [Hash Table Python Example](https://www.guru99.com/hash-table-data-structure.html#7)
* [Hash Table Code Explanation](https://www.guru99.com/hash-table-data-structure.html#8)
* [Python Dictionary Example](https://www.guru99.com/hash-table-data-structure.html#9)
* [Complexity Analysis](https://www.guru99.com/hash-table-data-structure.html#10)
* [Real-world Applications](https://www.guru99.com/hash-table-data-structure.html#11)
* [Advantages of hash tables](https://www.guru99.com/hash-table-data-structure.html#12)
* [Disadvantages of hash tables](https://www.guru99.com/hash-table-data-structure.html#13)

**What is a Hash Table?**

A **HASH TABLE** is a data structure that stores values using a pair of keys and values. Each value is assigned a unique key that is generated using a hash function.

The name of the key is used to access its associated value. This makes searching for values in a hash table very fast, irrespective of the number of items in the hash table.

**Hash functions**

For example, if we want to store employee records, and each employee is uniquely identified using an employee number.

We can use the employee number as the key and assign the employee data as the value.

The above approach will require extra free space of the order of **(m \* n2)** where the variable m is the size of the array, and the variable n is the number of digits for the employee number. This approach introduces a storage space problem.

A hash function solves the above problem by getting the employee number and using it to generate a hash integer value, fixed digits, and optimizing storage space. The purpose of a hash function is to create a key that will be used to reference the value that we want to store. The function accepts the value to be saved then uses an algorithm to calculate the value of the key.

The following is an example of a simple hash function

h(k) = k1 % m

**HERE,**

* h(k) is the hash function that accepts a parameter k. The parameter k is the value that we want to calculate the key for.
* k1 % m is the algorithm for our hash function where k1 is the value we want to store, and m is the size of the list. We use the modulus operator to calculate the key.

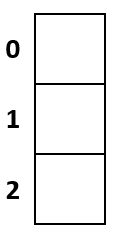
**Example**

Let’s assume that we have a list with a fixed size of 3 and the following values

[1,2,3]

We can use the above formula to calculate the positions that each value should occupy.

The following image shows the available indexes in our hash table.



**Step 1)**

Calculate the position that will be occupied by the first value like so

h(1) = 1 % 3

= 1

The value **1 will occupy** the space on **index 1**

**Step 2)**

Calculate the position that will be occupied by the second value

h(2) = 2 % 3

= 2

The value **2 will occupy** the space on **index 2**

**Step 3)**

Calculate the position that will be occupied by the third value.

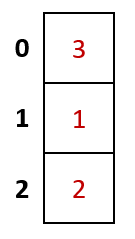
h(3) = 3 % 3

= 0

The value **3 will occupy** the space on **index 0**

**Final Result**

Our filled in hash table will now be as follows.



**Qualities of a good hash function**

A good hash function should have the following qualities.

* The formula for generating the hash should use the data’s value to be stored in the algorithm.
* The hash function should generate unique hash values even for input data that has the same amount.
* The function should minimize the number of collisions. Collisions occur when the same value is generated for more than one value.
* The values must be distributed consistently across the whole possible hashes.

**Collision**

A collision occurs when the algorithm generates the same hash for more than one value.

Let’s look at an example.

Suppose we have the following list of values

[3,2,9,11,7]

Let’s assume that the size of the hash table is 7, and we will use the formula (k1 % m) where m is the size of the hash table.

The following table shows the hash values that will be generated.

| **Key** | **Hash Algorithm (k1 % m)** | **Hash Value** |
| --- | --- | --- |
| 3 | 3 % 7 | 3 |
| 2 | 3 % 7 | 2 |
| 9 | 3 % 7 | 2 |
| 11 | 3 % 7 | 4 |
| 7 | 3 % 7 | 0 |

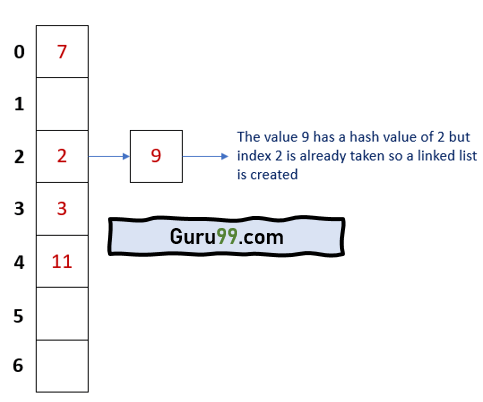
As we can see from the above results, the values 2 and 9 have the same hash value, and we cannot store more than one value at each position.

The given problem can be solved by either using chaining or probing. The following sections discuss chaining and probing in detail.

**Chaining**

Chaining is a technique that is used to solve the problem of collision by using linked lists that each have unique indexes.

The following image visualizes how a chained list looks like



Both 2 and 9 occupy the same index, but they are stored as linked lists. Each list has a unique identifier.

**Benefits of chained lists**

The following are the benefits of chained lists:

* Chained lists have better performance when inserting data because the order of insert is O(1).
* It is not necessary to resize a hash table that uses a chained list.
* It can easily accommodate a large number of values as long as free space is available.

**Probing**

The other technique that is used to resolve collision is probing. When using the probing method, if a collision occurs, we can simply move on and find an empty slot to store our value.

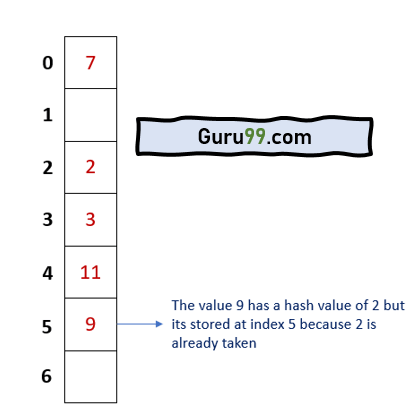
The following are the methods of probing:

Method Description

Linear probing Just like the name suggests, this method searches for empty slots linearly starting from the position where the collision occurred and moving forward. If the end of the list is reached and no empty slot is found. The probing starts at the beginning of the list.

Quadratic probing This method uses quadratic polynomial expressions to find the next available free slot.

Double Hashing This technique uses a secondary hash function algorithm to find the next free available slot.

Using our above example, the hash table after using probing would appear as follows:

**Hash table operations**

Here, are the Operations supported by Hash tables:

* **Insertion** – this Operation is used to add an element to the hash table
* **Searching** – this Operation is used to search for elements in the hash table using the key
* **Deleting** – this Operation is used to delete elements from the hash table

**Inserting data operation**

The insert operation is used to store values in the hash table. When a new value is stored in the hash table, it is assigned an index number. The index number is calculated using the hash function. The hash function resolves any collisions that occur when calculating the index number.

**Search for data operation**

The search operation is used to look-up values in the hash table using the index number. The search operation returns the value that is linked to the search index number. For example, if we store the value 6 at index 2, the search operation with index number 2 will return the value 6.

**Delete data operation**

The delete operation is used to remove a value from a hash table. To delete the Operation is done using the index number. Once a value has been deleted, the index number is made free. It can be used to store other values using the insert operation.

**Hash Table Implementation with Python Example**

Let’s look at a simple example that calculates the hash value of a key

def hash\_key( key, m):

return key % m

m = 7

print(f'The hash value for 3 is {hash\_key(3,m)}')

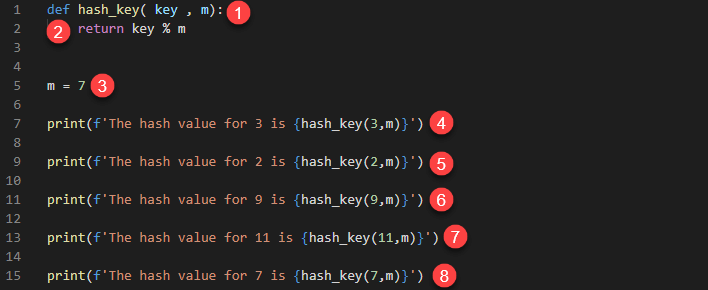
print(f'The hash value for 2 is {hash\_key(2,m)}')

print(f'The hash value for 9 is {hash\_key(9,m)}')

print(f'The hash value for 11 is {hash\_key(11,m)}')

print(f'The hash value for 7 is {hash\_key(7,m)}')

**Hash Table Code Explanation**



**HERE,**

1. Defines a function hash\_key that accepts the parameters key and m.
2. Uses a simple modulus operation to determine the hash value
3. Defines a variable m that is initialized to the value 7. This is the size of our hash table
4. Calculates and prints the hash value of 3
5. Calculates and prints the hash value of 2
6. Calculates and prints the hash value of 9
7. Calculates and prints the hash value of 11
8. Calculates and prints the hash value of 7

Executing the above code produces the following results.

The hash value for 3 is 3

The hash value for 2 is 2

The hash value for 9 is 2

The hash value for 11 is 4

The hash value for 7 is 0

**Python Dictionary Example**

Python comes with a built-in data type called Dictionary. A dictionary is an example of a hash table. It stores values using a pair of keys and values. The hash values are automatically generated for us, and any collisions are resolved for us in the background.

The following example shows how you can use a dictionary data type in python 3

employee = {

'name': 'John Doe',

'age': 36,

'position': 'Business Manager.'

}

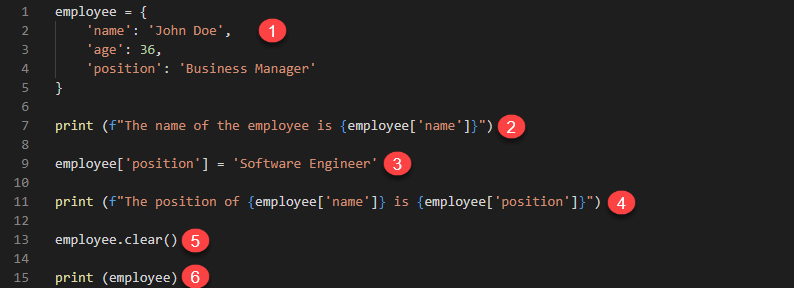
print (f"The name of the employee is {employee['name']}")

employee['position'] = 'Software Engineer'

print (f"The position of {employee['name']} is {employee['position']}")

employee.clear()

print (employee)



**HERE,**

1. Defines a dictionary variable employee. The key name is used to store the value John Doe, age stores 36, and position stores the value Business Manager.
2. Retrieves the value of the key name and prints it in the terminal
3. Updates the value of the key position to the value Software Engineer
4. Prints the values of the keys name and position
5. Deletes all the values that are stored in our dictionary variable employee
6. Prints the value of employee

Running the above code produces the following results.

The name of the employee is John Doe.

The position of John Doe is a Software Engineer.

{}

**Complexity Analysis**

Hash tables have an average time complexity of O (1) in the best-case scenario. The worst-case time complexity is O(n). The worst-case scenario occurs when many values generate the same hash key, and we have to resolve the collision by probing.

**Real-world Applications**

In the real-world, hash tables are used to store data for

* Databases
* Associative arrays
* Sets
* Memory cache

**Advantages of hash tables**

Here, are pros/benefits of using hash tables:

* Hash tables have high performance when looking up data, inserting, and deleting existing values.
* The time complexity for hash tables is constant regardless of the number of items in the table.
* They perform very well even when working with large datasets.

**Disadvantages of hash tables**

Here, are cons of using hash tables:

* You cannot use a null value as a key.
* Collisions cannot be avoided when generating keys using. hash functions. Collisions occur when a key that is already in use is generated.
* If the hashing function has many collisions, this can lead to performance decrease.

**Summary:**

* Hash tables are used to store data using a pair of keys and values.
* A hash function uses a mathematical algorithm to calculate the hash value.
* A collision occurs when the same hash value is generated for more than one value.
* Chaining solves collision by creating linked lists.
* Probing solves collision by finding empty slots in the hash table.
* Linear probing searches for the next free slot to store the value starting from the slot where the collision occurred.
* Quadratic probing uses polynomial expressions to find the next free slot when a collision occurs.
* Double hashing uses a secondary hash function algorithm to find the next free slot when a collision occurs.
* Hash tables have better performance when compared to other data structures.
* The average time complexity of hash tables is O (1)
* A dictionary data type in python is an example of a hash table.
* Hash tables support insert, search and delete operations.
* A null value cannot be used as an index value.
* Collisions cannot be avoided in hash functions. A good hash function minimizes the number of collisions that occur to improve the performance.