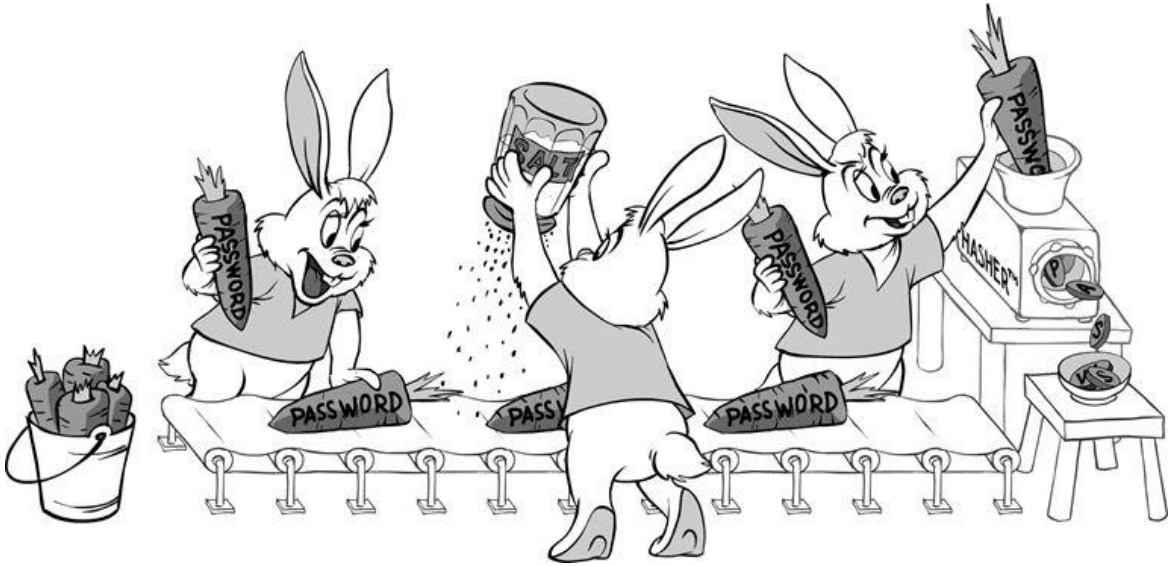


HASHING CONCEPT

Basics

Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects. Some examples of how hashing is used in our lives include:

- In universities, each student is assigned a unique roll number that can be used to retrieve information about them.
- In libraries, each book is assigned a unique number that can be used to determine information about the book, such as its exact position in the library or the users it has been issued to etc.

In both previous examples the students and books were hashed to a unique number.

Assume that you have an object and you want to assign a key to it to make searching easy. To store the **key/value pair,** you can use a simple array like a data structure where **keys (integers) can be used directly as an index to store values**. However, **in cases where the keys are large and cannot be used directly as an index, you should use *hashing*.**

In hashing, large keys are converted into small keys by using hash functions. The values are then stored in a data structure called hash table.

The idea of hashing is to distribute **entries (key/value pairs)** uniformly across an array. Each element is assigned a key (converted key). **By using that key you can access the element in O(1) time**. Using the key, the algorithm (hash function) computes an index that suggests where an entry can be found or inserted.

Hashing is implemented in two steps:

1. **An element is converted into an integer by using a hash function.** This element can be used as an index to store the original element, which falls into the hash table.
2. The element is stored in the hash table where it can be quickly retrieved using hashed key.
   hash = hashfunc(key)
   index = hash % array_size

In this method, the hash is independent of the array size and it is then reduced to an index (a number between 0 and array_size − 1) by using the modulo operator (%).

Hash function

A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes.

To achieve a good hashing mechanism, It is important to have a good hash function with the following basic requirements:

1. Easy to compute: It should be easy to compute and must not become an algorithm in itself.
2. Uniform distribution: It should provide a uniform distribution across the hash table and should not result in clustering.
3. Less collisions: Collisions occur when pairs of elements are mapped to the same hash value. These should be avoided.

Need for a good hash function (Good Example)

Assume that you have to store strings in the hash table by using the hashing technique {“abcdef”, “bcdefa”, “cdefab” , “defabc” }.

To compute the index for storing the strings, use a hash function that states the following:

1. The index for a specific string will be equal to the sum of the ASCII values of the characters modulo 599.
2. As 599 is a prime number, it will reduce the possibility of indexing different strings (collisions). It is recommended that you use prime numbers in case of modulo. The ASCII values of a, b, c, d, e, and f are 97, 98, 99, 100, 101, and 102 respectively. Since all the strings contain the same characters with different permutations, the sum will 599.
3. The hash function will compute the same index for all the strings and the strings will be stored in the hash table in the following format. As the index of all the strings is the same, you can create a list on that index and insert all the strings in that list.


Hash Table
Here all strings are sorted at same index
Index
0
1
2
abcdef
bcdefa
cdefab
defabc
3
4
-
-
-

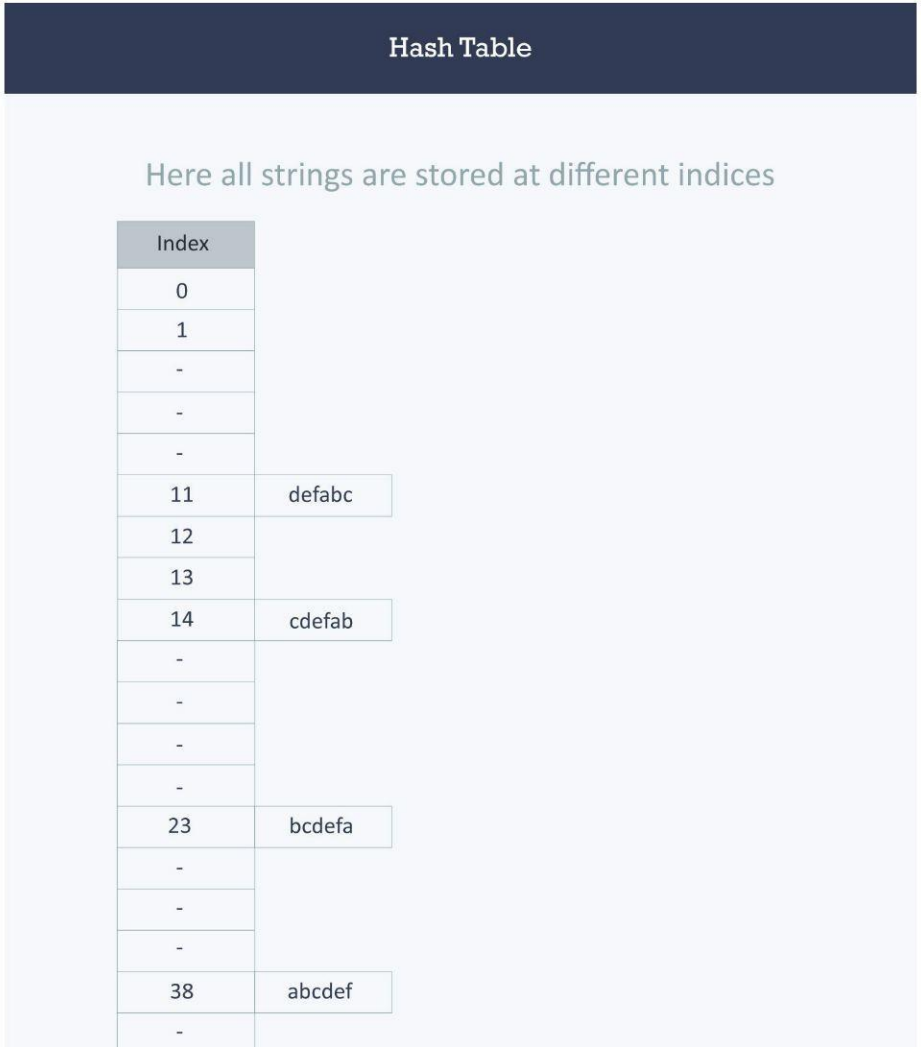
HASH TABLE

Here, **it will take O(n) time (where n is the number of strings) to access a specific string.** This shows that the hash function is not a good hash function.

Let’s try a different hash function. The index for a specific string will be equal to sum of ASCII values of characters multiplied by their respective order in the string after which it is modulo with 2069 (prime number).

String	Hash function	Index
abcdef	(97*1* + *98*2 + 99*3* + *100*4 + 101*5* + *102*6)%2069	38
bcdefa	(98*1* + *99*2 + 100*3* + *101*4 + 102*5* + *97*6)%2069	23
cdefab	(99*1* + *100*2 + 101*3* + *102*4 + 97*5* + *98*6)%2069	14
defabc	(100*1* + *101*2 + 102*3* + *97*4 + 98*5* + *99*6)%2069	11

HASH TABLE (OPTIMIZED)



Hash table

A **hash table is a data structure** that is used to **store keys/value pairs**.

It uses a hash function to **compute an index** into an array in which an element will be inserted or searched.

By using a good hash function, hashing can work well.

Under reasonable assumptions, the average time required to search for an element in a **hash table is O(1)**.

STATIC AND DYNAMIC HASHING

The **main difference** between static and dynamic hashing is that, **in static hashing, the resultant data bucket address is always the same while, in dynamic hashing, the data buckets grow or shrink according to the increase and decrease of records.**

Hashing uses mathematical functions called hash functions to generate addresses of data records. In addition, the memory locations that store data are called data buckets. There are two types of hashing called static and dynamic hashing.

What is Static Hashing

In static hashing, the resultant data bucket address is always the same. In other words, the bucket address does not change. Thus, in this method, the number of data buckets in memory remains constant throughout.

Operations of static hashing are as follows.

Insertion – When entering a record using static hashing, the hash function (h) calculates the bucket address for search key (k), where the record will be stored. Bucket address = h(K).

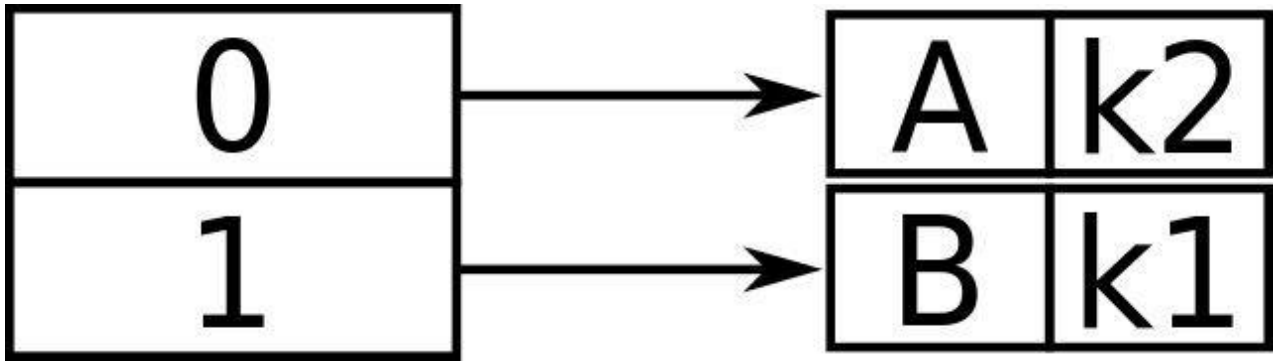
Search – When obtaining a record, the same hash function helps to obtain the address of the bucket where the data is stored.

Delete – After fetching the record, it is possible to delete the records for that address in memory.

Update – After searching the record using a hash function, it is possible to update that record.

What is Dynamic Hashing

An issue in static hashing is bucket overflow. Dynamic hashing helps to overcome this issue. It is also called **Extendable hashing method**. In this method, the data buckets increase and decrease depending on the number of records. It allows performing operations such as insertion, deletion etc. without affecting the performance.



STATIC HASHING
VERSUS
DYNAMIC HASHING

STATIC HASHING	DYNAMIC HASHING
A hashing technique that allows users to perform lookups on a finalized dictionary set (all objects in the dictionary are final and not changing)	A hashing technique in which the data buckets are added and removed dynamically and on demand
Resultant data bucket address is always the same	Data buckets change depending on the records
Less efficient	More efficient