CSE-213
(Data Structure)

Lecture on

Hashing

Md. Jalal Uddin

Lecturer

Department of CSE

City University

Email: jalalruice@gmail.com

No: 01717011128 (Emergency Call)

Department of Computer Science & Engineering (CSE)
City University, Khagan, Birulia, Savar, Dhaka-1216, Bangladesh

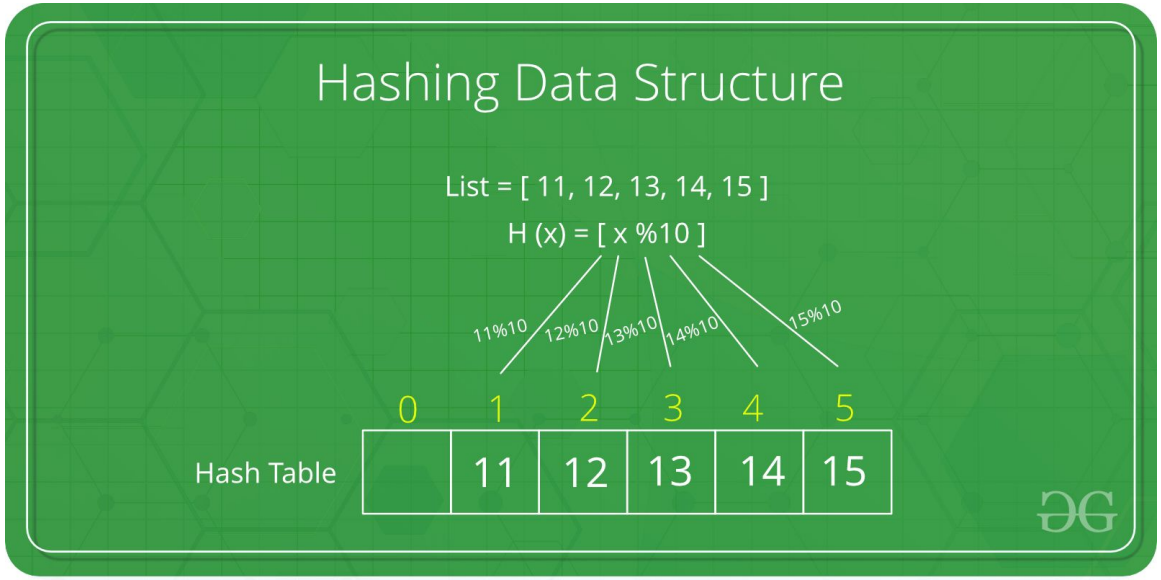
What is Hashing Data

Hashing is a technique used in data structures to store and retrieve data efficiently. It involves using a hash function to map data items to a fixed-size array which is called a hash table.

Hashing involves mapping data to a unique value, called a hash code. The hash code is then used to index into an array, where the data is stored. To retrieve the data, the hash code is simply re-computed and used to index into the array.

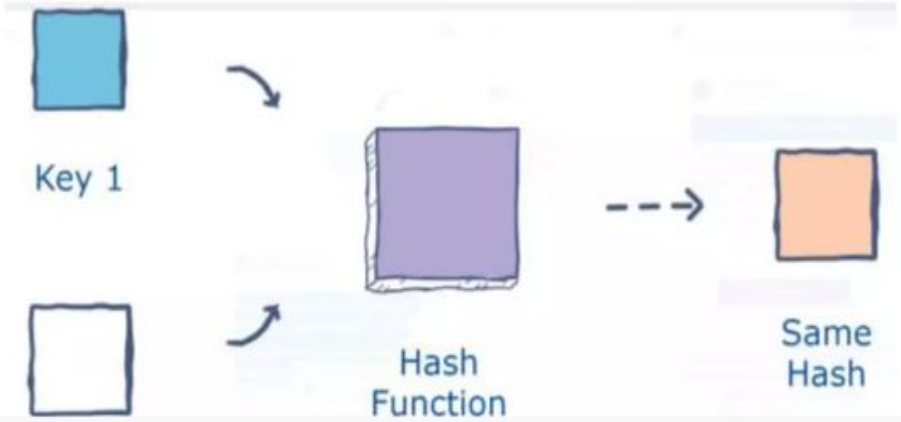
Hashing (intro)

It is a Technique or process of mapping keys , value into the table by using a hash function. It is done for faster a access to elements . The efficiency of mapping depends on the efficiency of the hash function used.



COLLISION RESOLUTION TECHNIQUES

- What is Collision?
- Keep in mind that two keys can generate the same hash. This phenomenon is known as a collision.
- Since 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 and must be handled using some collision handling technique.



COLLISION RESOLUTION TECHNIQUES (CONTD.)

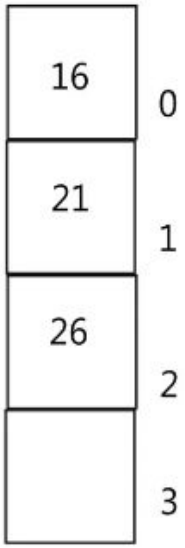
E.g. Consider the following keys: (16,21,26,20)

Size of the Hash Table is 3.

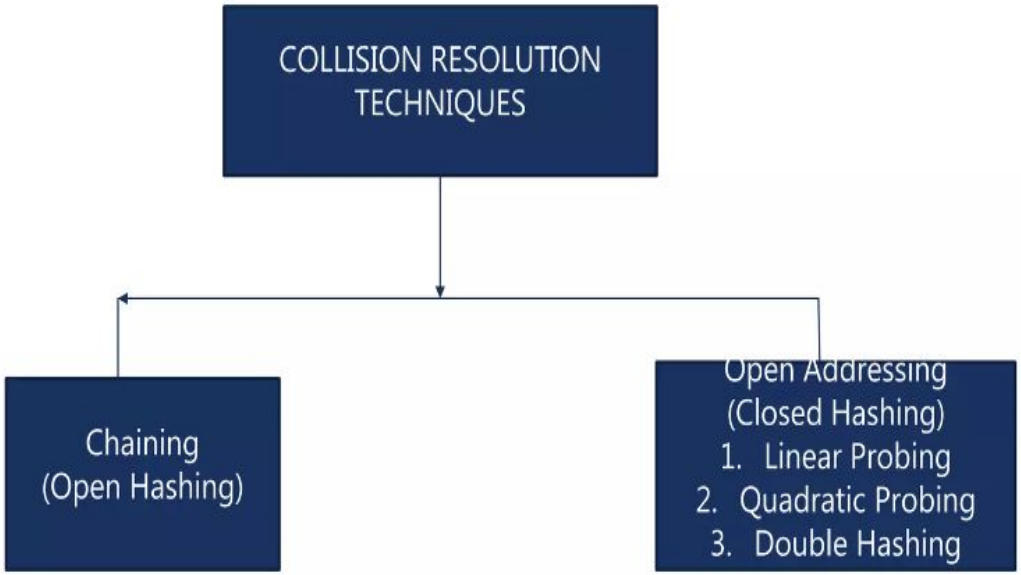
Now we will start inserting the keys.

For 16 = 16 % 4= 0
21 = 21% 4 = 1
26 = 26 % 4 = 2
20 = 20 % 4 = 0

For the last key (20), there is already an element where 20 is to be inserted,
This is where Collision Resolution Technique comes into picture.



COLLISION RESOLUTION TECHNIQUES (CONTD.)



linear probing

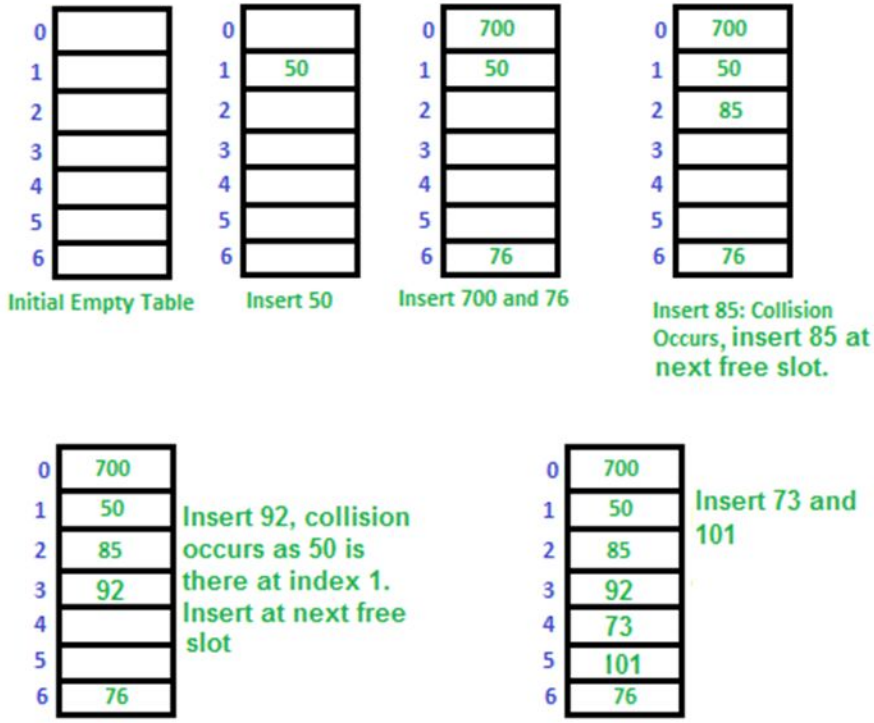
In linear probing, the hash table is searched sequentially that starts from the original location of the hash. If in case the location that we get is already occupied, then we check for the next location.

- *Let **hash(x)** be the slot index computed using a hash function and **S** be the table size*
- *If slot hash(x) % S is full, then we try (hash(x) + 1) % S If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S*
- *If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S*

linear probing

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

which means hash(key)= key% S, here S=size of the table =7,indexed from 0 to 6.We can define the hash function as per our choice if we want to create a hash table , although it is fixed internally with a pre-defined formula.



PROS AND CONS:

- The **advantages** of linear probing are as follows –

Linear probing requires very less memory.

It is less complex and is simpler to implement.

- The **disadvantages** of linear probing are as follows –

Linear probing causes a scenario called "primary clustering" in which there are large blocks of occupied cells within the hash table.

The values in linear probing tend to cluster which makes the probe sequence longer and lengthier.

RESULT & DISCUSSION:

- Hashing is most commonly used to implement *hash tables*. A hash table stores key/value pairs in the form of a list where any element can be accessed using its index.
- Since there is no limit to the number of key/value pairs, we can use a hash function to map the keys to the size of the table; the hash value becomes the index for a given element.
- A simple hashing approach would be to take the modular of the key (assuming it's numerical) against the table's size:
- index = key \text{ } MOD \text{ } tableSize*index= key MOD tableSize*
- This will ensure that the hash is always within the limits of the table size.

CONCLUSION:

- 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.