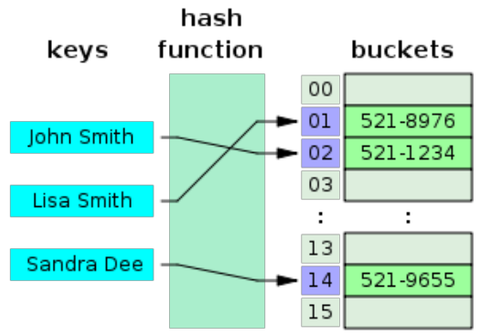
Hash Table

- Open Hashing
- Closed Hashing
  - Linear probing
  - Quadratic probing
  - Double hashing

Hash Table

- It is used to store (Key,Value) pairs.
- Each key corresponds to only one value. Some uses: Compilers, databases, search engines In cases where the number of recordings is high, its speed becomes evident.
- Balanced trees like BST, AVL: $O(\log_2 n)$
- Hash table: O(1)
- Operations in the hash table usually require constant time.
- This is not preferred in cases where the number of records is low due to the processing load. The hash table can be used if the table size is large enough and sorting is not required.



Hash table

Type	Unordered associative array
Invented	1953

Time complexity in big O notation

Algorithm	Average	Worst case
Space	O(*n*)[1]	O(*n*)
Search	O(1)	O(*n*)
Insert	O(1)	O(*n*)
Delete	O(1)	O(*n*)

Kaynak: https://en.wikipedia.org/wiki/Hash_table

Hash Function

- hash key = key % lengthThe hash table uses a hash function to store data into an array.
- The hash function generates an index based on the key. Array elements are accessed by this index. If the pointed position is NULL, the searched key is not in the table.
- The key value is an integer.
- However, key can also be a string expression. In this case, an integer as follows is obtained from the string expression first.
- for(int j = 0; j < stringKey.length(); j++)

```
        hashValue += stringKey[j];
int hashKey = hashValue % tableSize;
```

Assume a table with 8 slots:

Hash key = key % table size

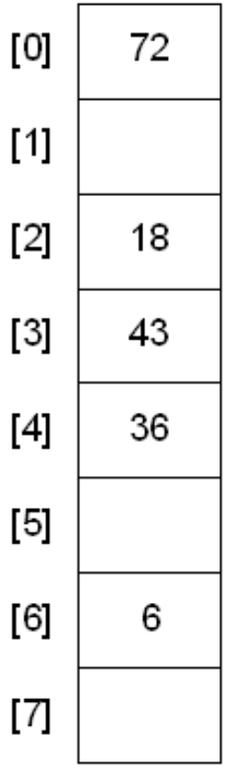
4 = 36 % 8

2 = 18 % 8

0 = 72 % 8

3 = 43 % 8

6 = 6 % 8



Hash collision

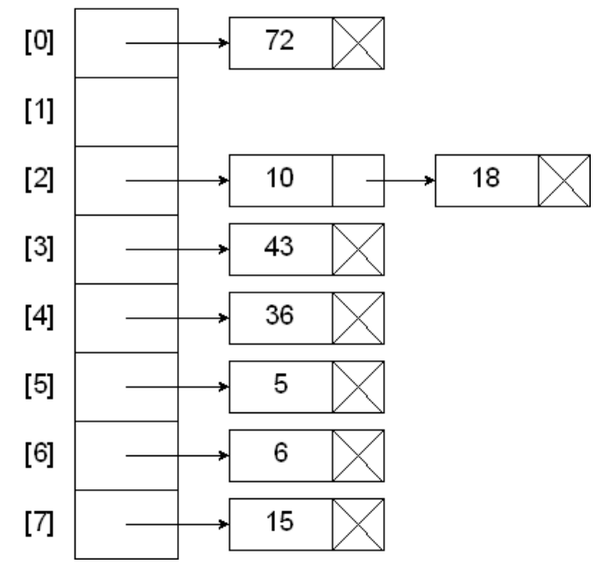
- Different key values can use the same location on the array, producing the same hash values.
- For example, let the key=55 and the table size=21.
- In this case hash key= 55%21= 13 is obtained.
- If key=34, the same location is selected. hash key= 34%21= 13 is obtained.
- Approaches used to prevent collisions:
  - **Open Hashing (separate chaining)**
  - **Closed Hashing (open addressing)**

Open Hashing (separate chaining)

- Each of the table elements is thought of as a one-way linked list.
- It is desirable to have a node at each location.
- However, in the case of a hash collision, the new element is added to the beginning of the list and the list begins to grow.
- Increasing the number of collisions causes the list sizes to increase and the performance to decrease due to the need for search operations.

Hash key = key % table size

```
4    = 36 % 8
2    = 18 % 8
0    = 72 % 8
3    = 43 % 8
6    =  6 % 8
2    = 10 % 8
5    =  5 % 8
7    = 15 % 8
```



http://faculty.cs.niu.edu/~freedman/340/340notes/340hash.htm

https://www.cs.usfca.edu/~galles/visualization/OpenHash.html

Closed Hashing (open addressing)

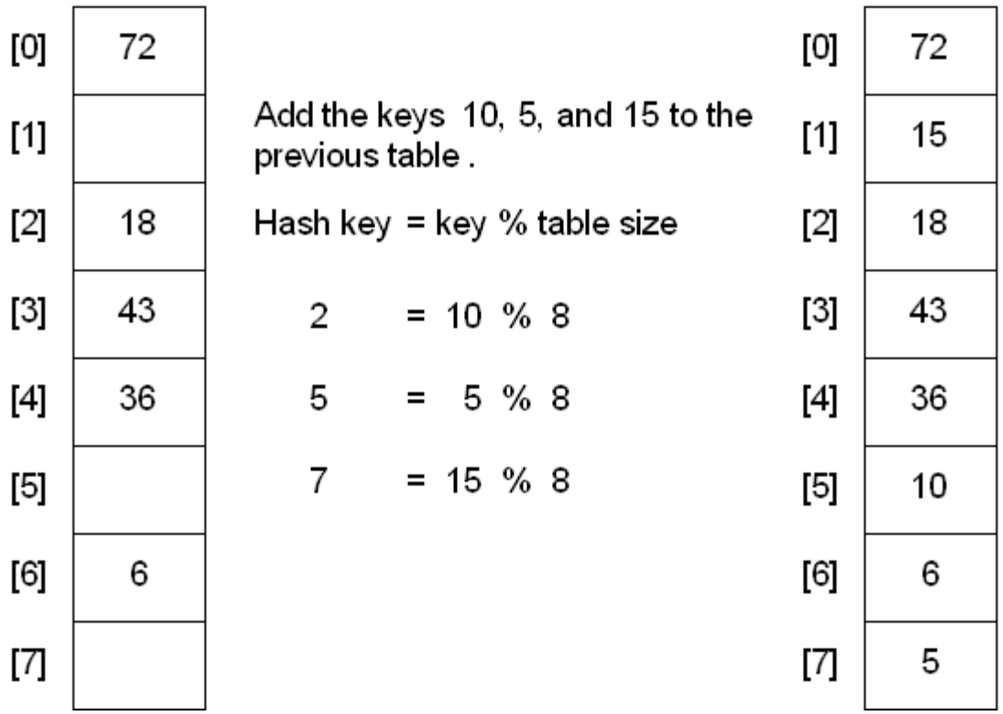
- If the position on the table is full, the next one is selected.
- The following approaches are generally used to select the new location.
- Linear probing
- Quadratic probing
- Double hashing

Linear probing

- If the position is full, the next is selected. The remainder of the division by the table size is used so that the new location does not exceed the table size.
- Full rows can form large chains (Primary clustering)
- Table load factor:

λ =Number of elements/Table size

- λ <0.5 desirable for better performance
- When the element whose example key is 10 is added in the example table, hash key=2 is obtained. Because this location and the following two locations are occupied, location 5 is selected. Similarly, keys 5 and 15 are added.
- Here, long chains are formed because the table load factor is high.



http://faculty.cs.niu.edu/~freedman/340/340notes/340hash.htm

Double hashing

- The location is calculated as before.
- $hash_1$ (key) = key % MAX
- If the location is full, the new location is selected with another hash function as follows, reducing the possibility of further collisions.
- $h_i(key)=(hash_1(key) + i^* hash_2(key)) \%MAX$
- $hash_2(key)$ = R - (key % R)
- R:A prime number close to the table size.
- R<Tablesize.
- When the location is full, it is incremented by the hash2 value each time.

Table Size = 10 elements
$Hash_1(key)$ = key % 10
$Hash_2(key)$ = 7 – (k % 7)

Insert keys : 89, 18, 49, 58, 69

Hash(89) = 89 % 10 = 9

Hash(18) = 18 % 10 = 8

Hash(49) = 49 % 10 = 9 a collision !
= 7 – (49 % 7)
= 7 positions from [9]

Hash(58) = 58 % 10 = 8
= 7 – (58 % 7)
= 5 positions from [8]

Hash(69) = 69 % 10 = 9
= 7 – (69 % 7)
= 1 position from [9]

[0]	69
[1]	
[2]	
[3]	58
[4]	
[5]	
[6]	49
[7]	
[8]	18
[9]	89