

Hashing, Hash Function and Hash Table

What is Hashing

- Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects.
- Examples of how hashing is used in our lives:
 - 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
- **Hashing** is the solution that can be used in almost all such situations and worst case complexity of hashing is still $O(n)$ (better than BST which is $O(\log n)$), but it gives $O(1)$ on the average.

What is Hashing (Cond...)

- Here we use ***hash function*** that converts a given input number or any other key to a smaller number and uses the small number as index in a table called ***hash table***.
- Hash Function
 - Hash function maps a big number or string to a small integer that can be used as index in hash table
- Hash Table
 - An array that stores pointers to records corresponding to a given input number

HashTable ADT

The common operations for hash table are:

- **CreatHashTable**: Creates a new hash table
- **HashSearch**: Searches the key in hash table
- **HashInsert**: Inserts a new key into hash table
- **HashDelete**: Deletes a key from hash table
- **DeleteHashTable**: Deletes the hash table

We can use array as a hash table.

For understanding the use of hash tables, let us consider the following example:

Give an algorithm for printing the first repeated character if there are duplicated elements in it. Let us think about the possible solutions.

The simple and brute force way of solving is: given a string, for each character check whether that character is repeated or not.

The time complexity of this approach is $O(n)$ with $O(1)$ space complexity.

Better Solution: worst case complexity = $O(n)$

```
char FirstRepeatedChar ( char *str ) {  
    int i, len=strlen(str);  
    int count[256]; //additional array  
    for(i=0; i<256; ++i)  
        count[i] = 0;  
    for(i=0; i<len; ++i) {  
        if(count[str[i]]==1) {  
            printf("%c", str[i]);  
            break;  
        }  
        else count[str[i]]++;  
    }  
    if(i==len)  
        printf("No Repeated Characters");  
    return 0;  
}
```

- Using simple arrays is not the correct choice for solving the problems where the possible keys are very big
- Creating a huge array and storing the counters is not possible.
- That means there are a set of universal keys and limited locations in the memory.
- To solve this problem we need to somehow map all these possible keys to the possible memory locations.
- The process of mapping the keys to locations is called **hashing**.

Components of Hashing

Hashing has four key components:

1) Hash Table

2) Hash Functions

3) Collisions

4) Collision Resolution Techniques

Hash Table

A Hash-Table is a data structure, which maps keys to values.

It is an array of size M to store objects references.

Hash table is a generalization of array. With an array, we store the element whose key is k at a position k of the array

Hash Function

A hash function is a function, which generates an index in a table for a given object.

A hash function which generate a unique index for every object is called the perfect hash function.

Most simple hash function

```
unsigned int Hash(int key, int tableSize)//division method
{
    unsigned int hashValue = 0;
    hashValue = key;
    return hashValue % tableSize;
}
```

There are many hash functions, but this is the minimum that it should do. Various hash generation logics will be added to this function to generate a better hash.

Characteristics of Good Hash Functions

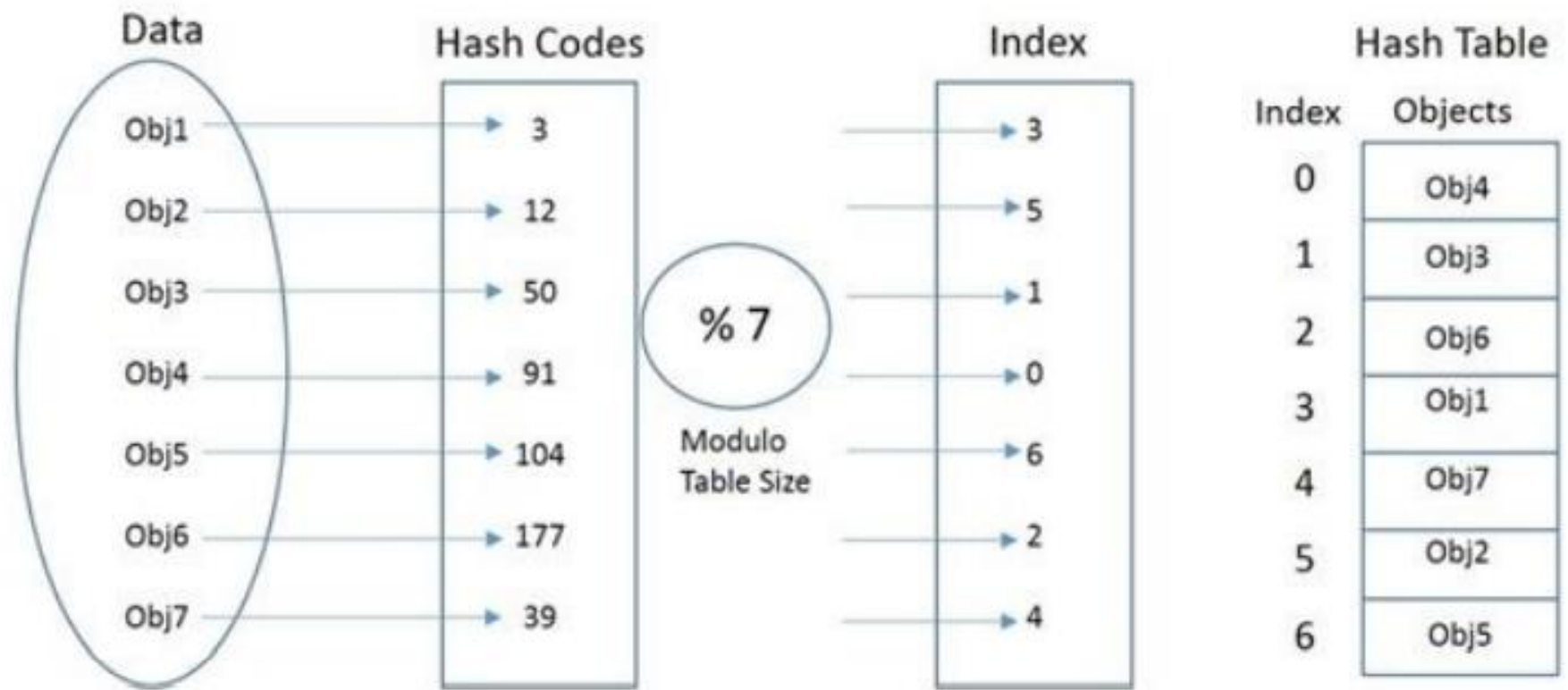
A good hash function should have the following characteristics:

- Minimize collision
- Be easy and quick to compute
- It should provide a uniform distribution of hash values
- Use all the information provided in the key
- Have a high load factor for a given set of keys

Load factor = Number of elements in Hash-Table / Hash-Table size

Hash Function

Object -----> Index



The process of **storing objects** using a hash function is as follows:

1. Create an array of size M to store objects, this array is called Hash-Table.
2. Find a hash code of an object by passing it through the hash function.
3. Take modulo of hash code by the size of Hash-table to get the index of the table where object will be stored.
4. Finally store the object in the designated index.

The process of **searching objects** in Hash-Table using a hash function is as follows:

1. Find a hash code of the object we are searching for by passing it through the hash function.
2. Take modulo of hash code by the size of Hash-table to get the index of the table where objects are stored.
3. Finally, retrieve the object from the designated index



0

1

2

3

4

5

6

7

8

9

10

Mia

M

77

i

105

a

97

279

4

--	--	--	--	--	--	--	--	--	--	--

0

1

2

3

4

5

6

7

8

9

10

Mia

M

77

i

105

a

97

279

4

				Mia						
0	1	2	3	4	5	6	7	8	9	10

Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1

	Tim			Mia						
0	1	2	3	4	5	6	7	8	9	10

Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5
Jan	J	74	a	97	n	110	281	6
Ada	A	65	d	100	a	97	262	9
Leo	L	76	e	101	o	111	288	2

Bea	Tim	Leo		Mia	Zoe	Jan			Ada	
0	1	2	3	4	5	6	7	8	9	10

Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5
Jan	J	74	a	97	n	110	281	6
Ada	A	65	d	100	a	97	262	9
Leo	L	76	e	101	o	111	288	2
Sam	S	83	a	97	m	109	289	3
Lou	L	76	o	111	u	117	304	7
Max	M	77	a	97	x	120	294	8
Ted	T	84	e	101	d	100	285	10

Bea	Tim	Leo	Sam	Mia	Zoe	Jan	Lou	Max	Ada	Ted
0	1	2	3	4	5	6	7	8	9	10

Index number = $\text{sum ASCII codes} \bmod \text{size of array}$

Bea	Tim	Leo	Sam	Mia	Zoe	Jan	Lou	Max	Ada	Ted
0	1	2	3	4	5	6	7	8	9	10

$$\text{Ada} = (65 + 100 + 97) = 262$$

Find Ada

$$262 \text{ Mod } 11 = 9$$

myData = Array(9)

Bea	Tim	Leo	Sam	Mia	Zoe	Jan	Lou	Max	Ada	Ted
0	1	2	3	4	5	6	7	8	9	10

Bea 27/01/1941 English Astronomer	Tim 08/06/1955 English inventor	Leo 31/12/1945 American Mathematician	Sam 27/04/1791 American inventor	Mia 20/02/1986 Russian Space Station	Zoe 19/06/1978 American Actress	Jan 13/02/1956 Polish Logician	Lou 27/12/1822 French Biologist	Max 23/04/1858 German Physicist	Ada 10/12/1815 English Mathematician	Ted 17/06/1937 American Philosopher
--	--	--	---	---	--	---	--	--	---	--

0

1

2

3

4

5

6

7

8

9

10

Understand Collision

Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5

Collision is the condition where two records needs to be stored in the same location, which is not possible.



Understand Collision

Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5
Sue	S	83	u	117	e	101	301	4

Bea	Tim			Mia	Zoe					
0	1	2	3	4	5	6	7	8	9	10

Collision Handling

- The process of finding an alternate location is called collision resolution
- Therefore, to maintain the performance of a hash table, it is important to manage collisions through various collision resolution techniques.
 - Open addressing
 - Linear Probing
 - Quadratic Probing
 - Double Hashing
 - Closed addressing

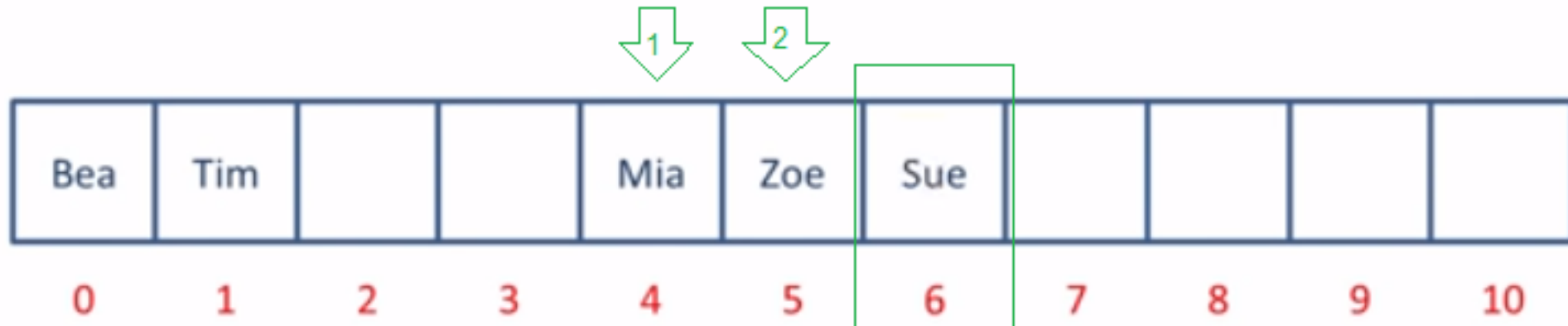
Linear Probing

- The function for rehashing is the following:
- $\text{rehash}(\text{key}) = (n + 1) \% \text{tablesize}$

One of the problems with linear probing is that table items tend to cluster together in the hash table. This means that the table contains groups of consecutively occupied locations that are called clustering.

Open Addressing (Linear)

Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5
Sue	S	83	u	117	e	101	301	4



Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5
Sue	S	83	u	117	e	101	301	4
Len	L	76	e	101	n	110	287	1



Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5
Sue	S	83	u	117	e	101	301	4
Len	L	76	e	101	n	110	287	1
Moe	M	77	o	111	e	101	289	3
Lou	L	76	o	111	u	117	304	7
Rae	R	82	a	97	e	101	280	5
Max	M	77	a	97	x	120	294	8
Tod	T	84	o	111	d	100	295	9

Bea	Tim	Len	Moe	Mia	Zoe	Sue	Lou	Rae	Max	Tod
0	1	2	3	4	5	6	7	8	9	10

Find Rae

$$\text{Rae} = (82 + 97 + 101) = 280$$

$$280 \text{ Mod } 11 = 5$$

myData = Array(5)

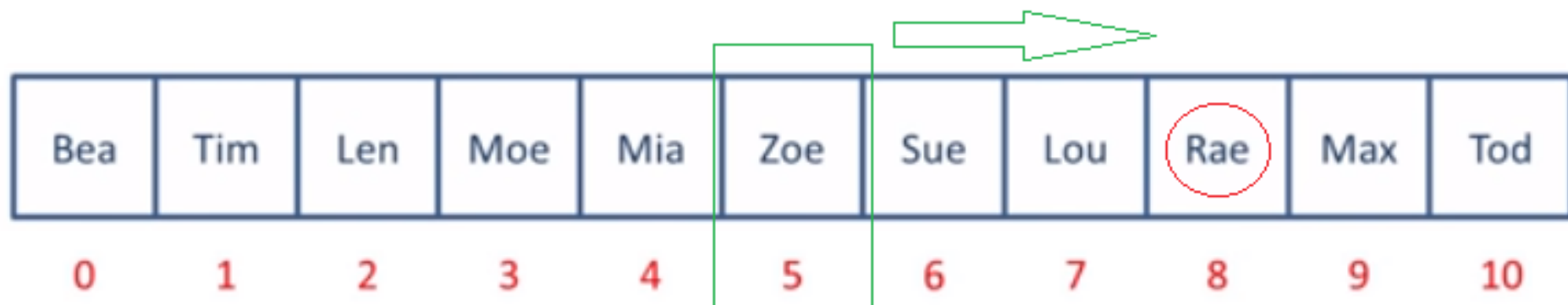
Bea	Tim	Len	Moe	Mia	Zoe	Sue	Lou	Rae	Max	Tod
0	1	2	3	4	5	6	7	8	9	10

Find Rae

$$\text{Rae} = (82 + 97 + 101) = 280$$

$$280 \bmod 11 = 5$$

myData = Array(5)



Bea	Tim	Len	Moe	Mia	Zoe	Sue	Lou	Rae	Max	Tod
0	1	2	3	4	5	6	7	8	9	10

Quadratic Probing

- The interval between probes increases proportionally to the hash value
- The problem of Clustering can be eliminated if we use the quadratic probing method.
- In quadratic probing, we start from the original hash location i . If a location is occupied, we check the locations $i + 1^2$, $i + 2^2$, $i + 3^2$, $i + 4^2$
- The function for rehashing is the following:

$$\text{rehash}(\text{key}) = (n + k^2) \% \text{tablesize}$$

Double Hashing

- The interval between probes is computed by another hash function. Double hashing reduces clustering in a better way.

The second hash function $h2$ should be: $h2(key) \neq 0$ and $h2 \neq h1$

We first probe the location $h1(key)$.

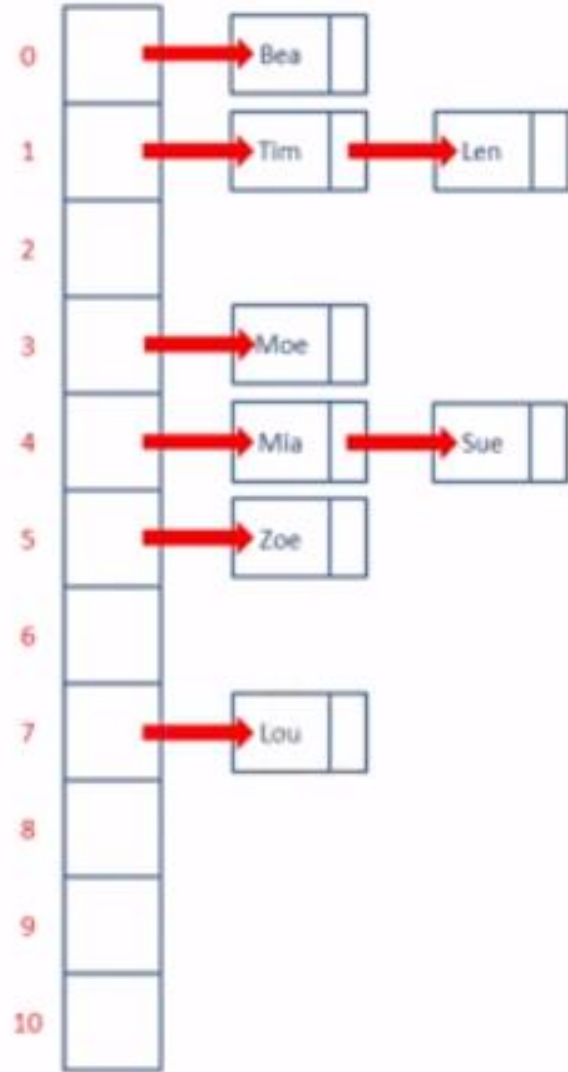
If the location is occupied, we probe the location $h1(key) + h2(key)$, $h1(key) + 2 * h2(key)$, ...

Example: Table size is 11 (0..10) Hash Function: assume $h1(key) = key \bmod 11$ and $h2(key) = 7 - (key \bmod 7)$

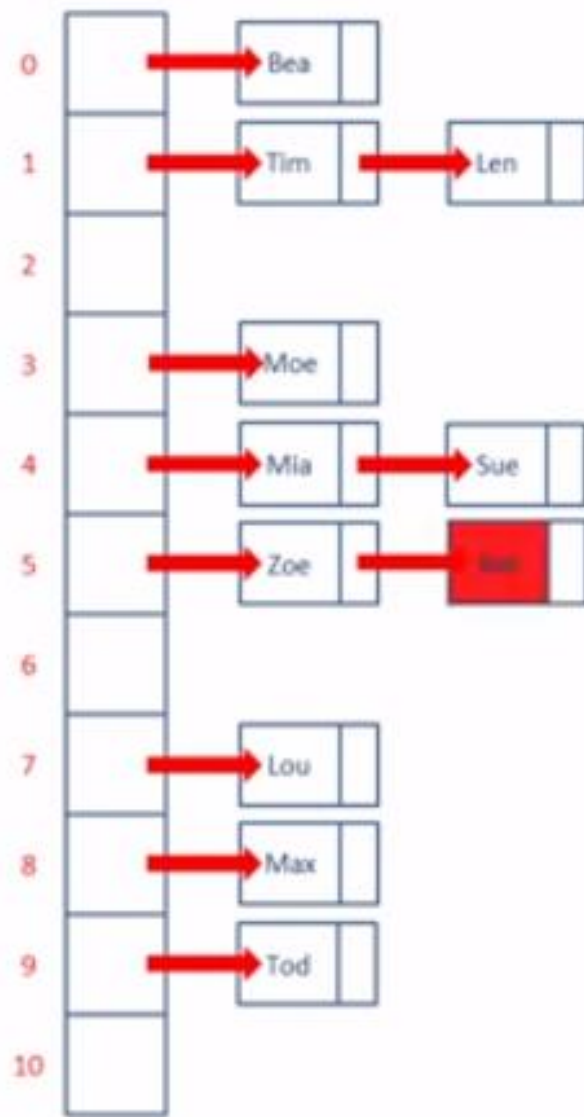
Closed Addressing (Non-Linear)



Mia M 77 i 105 a 97 279 4



Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5
Sue	S	83	u	117	e	101	301	4
Len	L	76	e	101	n	110	287	1
Moe	M	77	o	111	e	101	289	3
Lou	L	76	o	111	u	117	304	7



Find Rae $280 \text{ Mod } 11 = 5$

`myData = Array(5)`

Objectives of Hash Function

- Minimize collisions
- Uniform distribution of hash values
- Easy to calculate
- Resolve any collisions