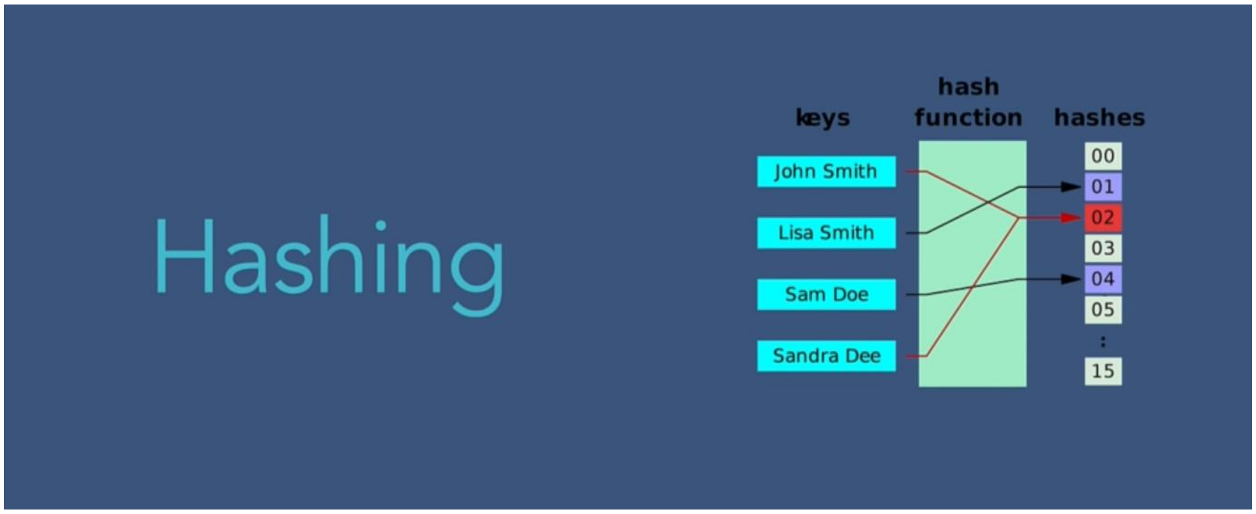
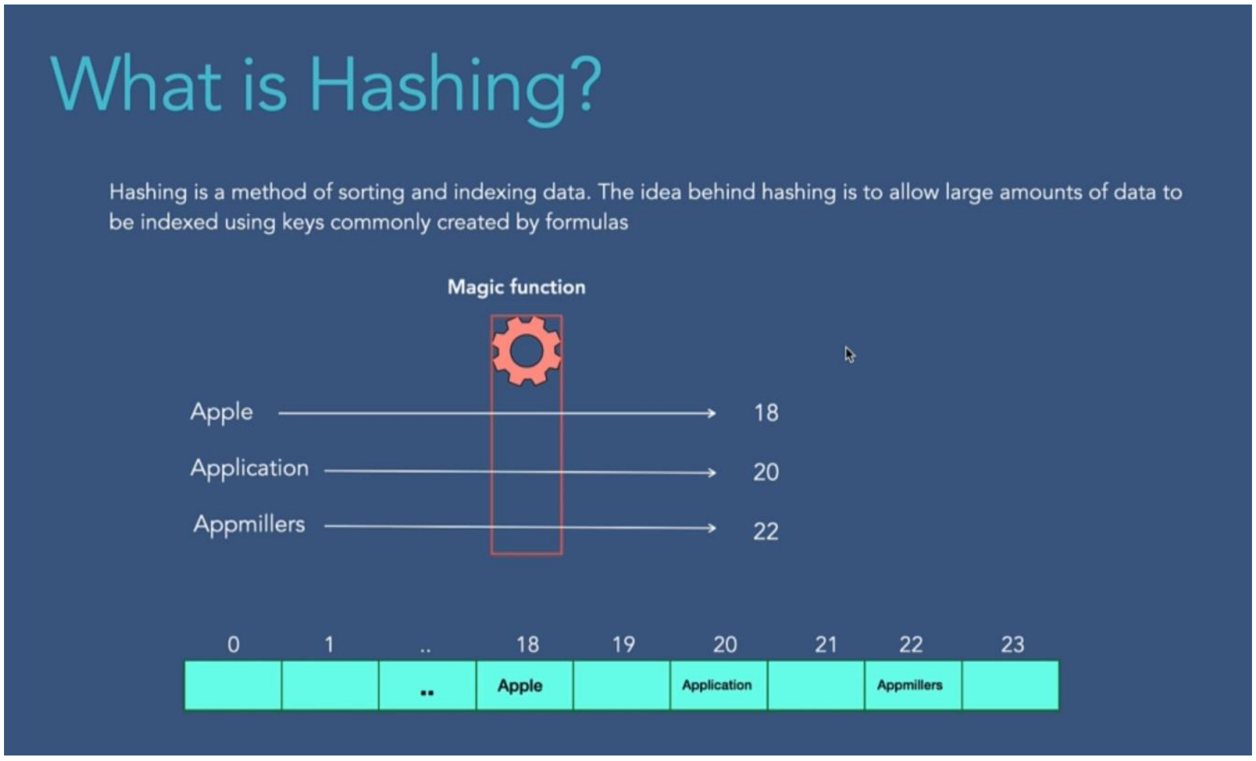
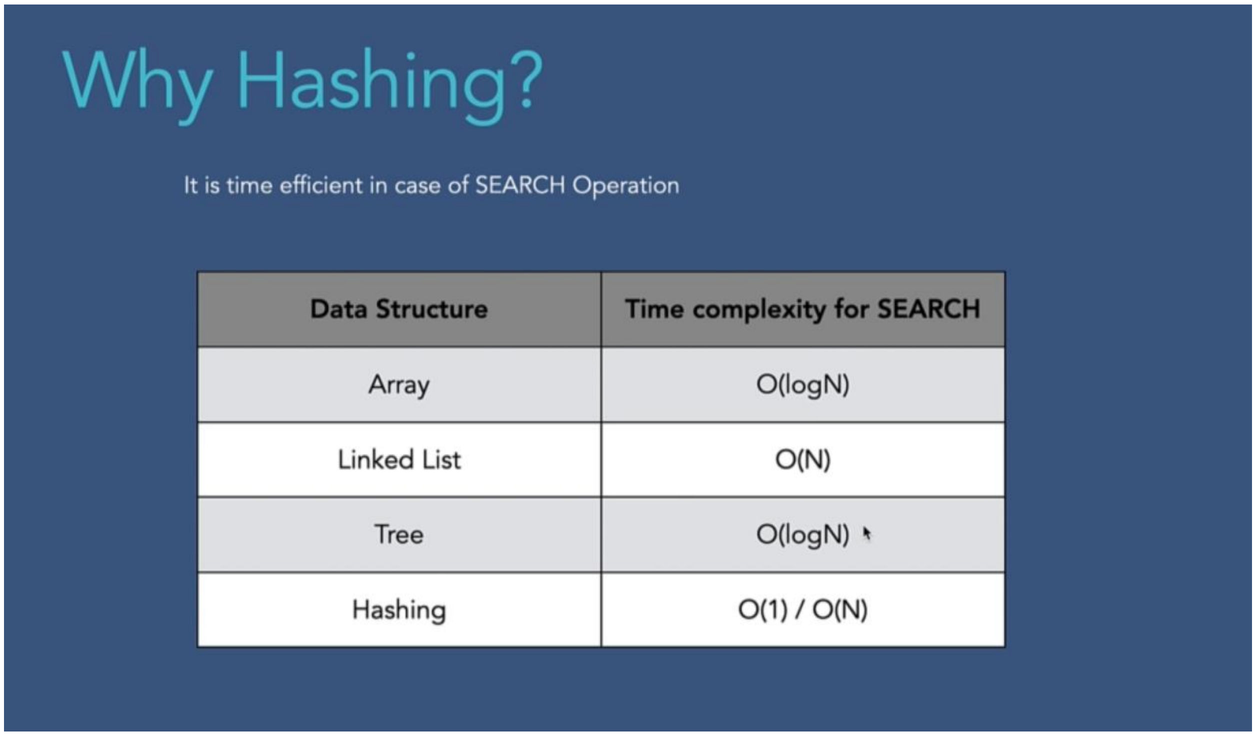




What is Hashing?

Hashing is a method of sorting and indexing data. The idea behind hashing is to allow large amounts of data to be indexed using keys commonly created by formulas

Hash implementation using Arrays

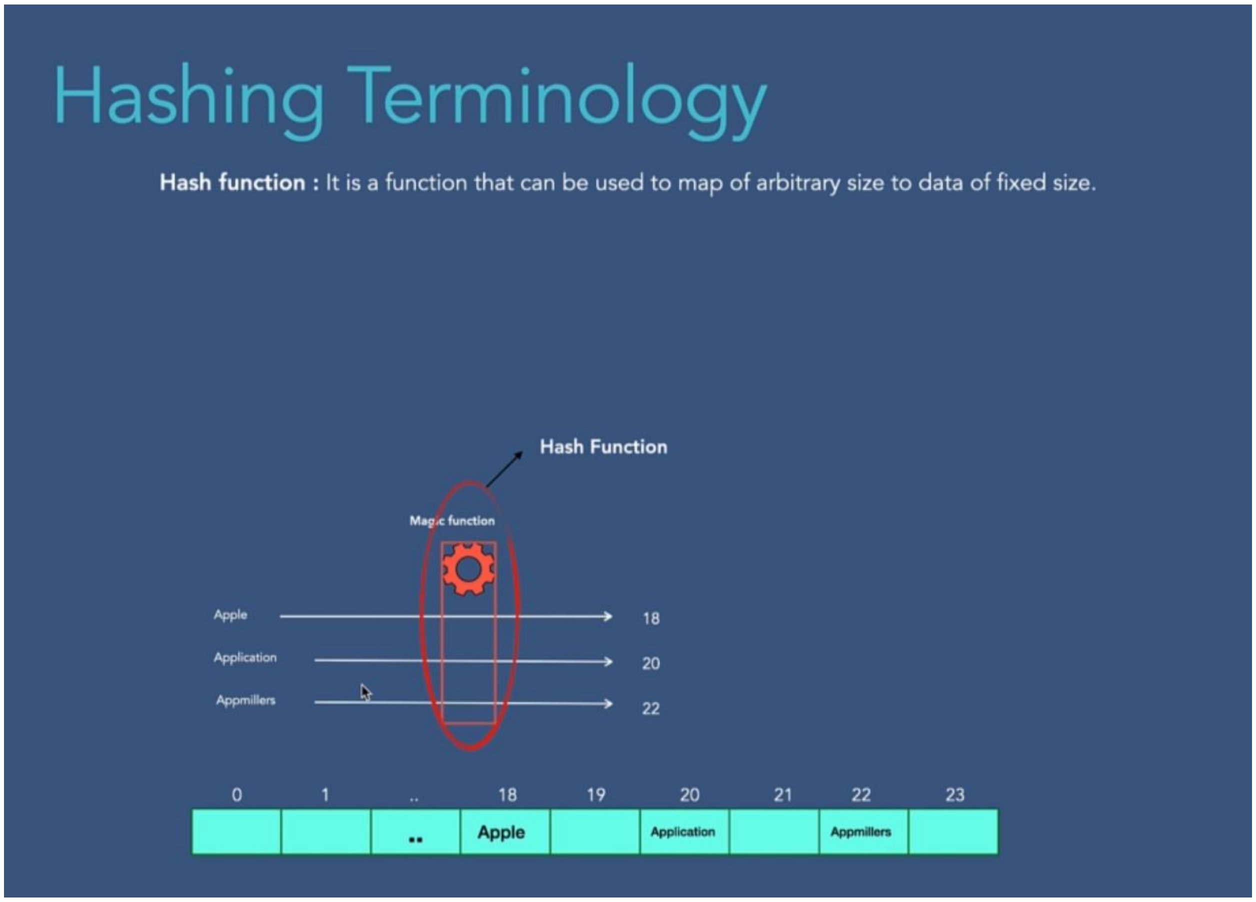


Why Hashing?

It is time efficient in case of SEARCH Operation

Data Structure	Time complexity for SEARCH
Array	O(logN)
Linked List	O(N)
Tree	O(logN)
Hashing	O(1) / O(N)

Best Time complexities of Data Structures



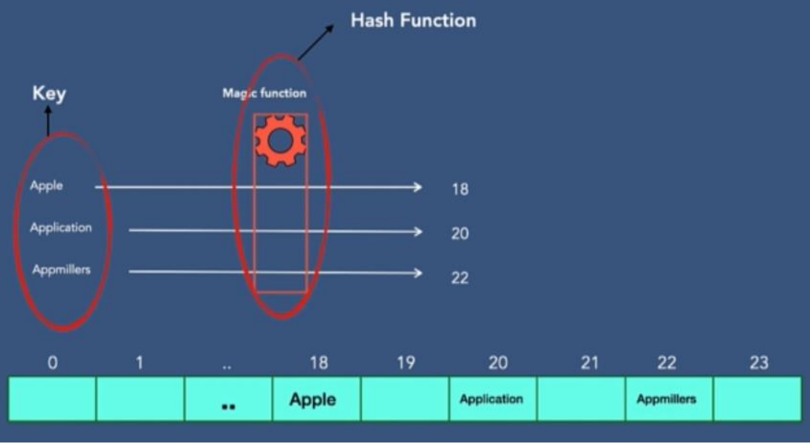
Hashing Terminology

Hash function : It is a function that can be used to map of arbitrary size to data of fixed size.

Hashing Terminology

Hash function : It is a function that can be used to map of arbitrary size to data of fixed size.

Key : Input data by a user

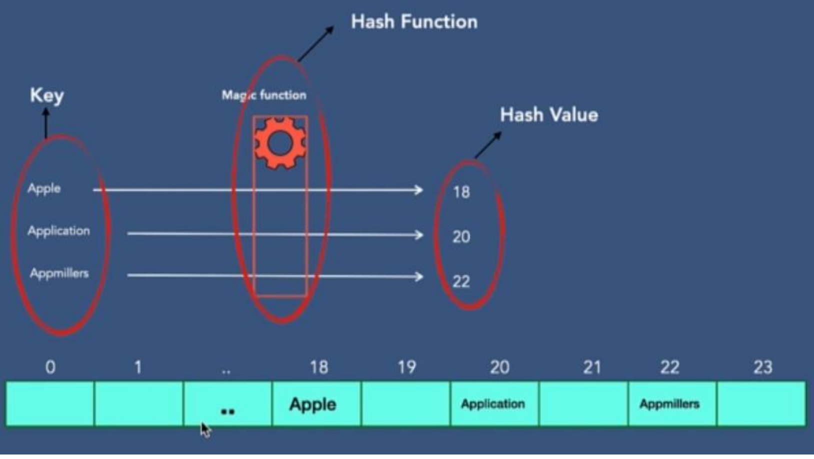


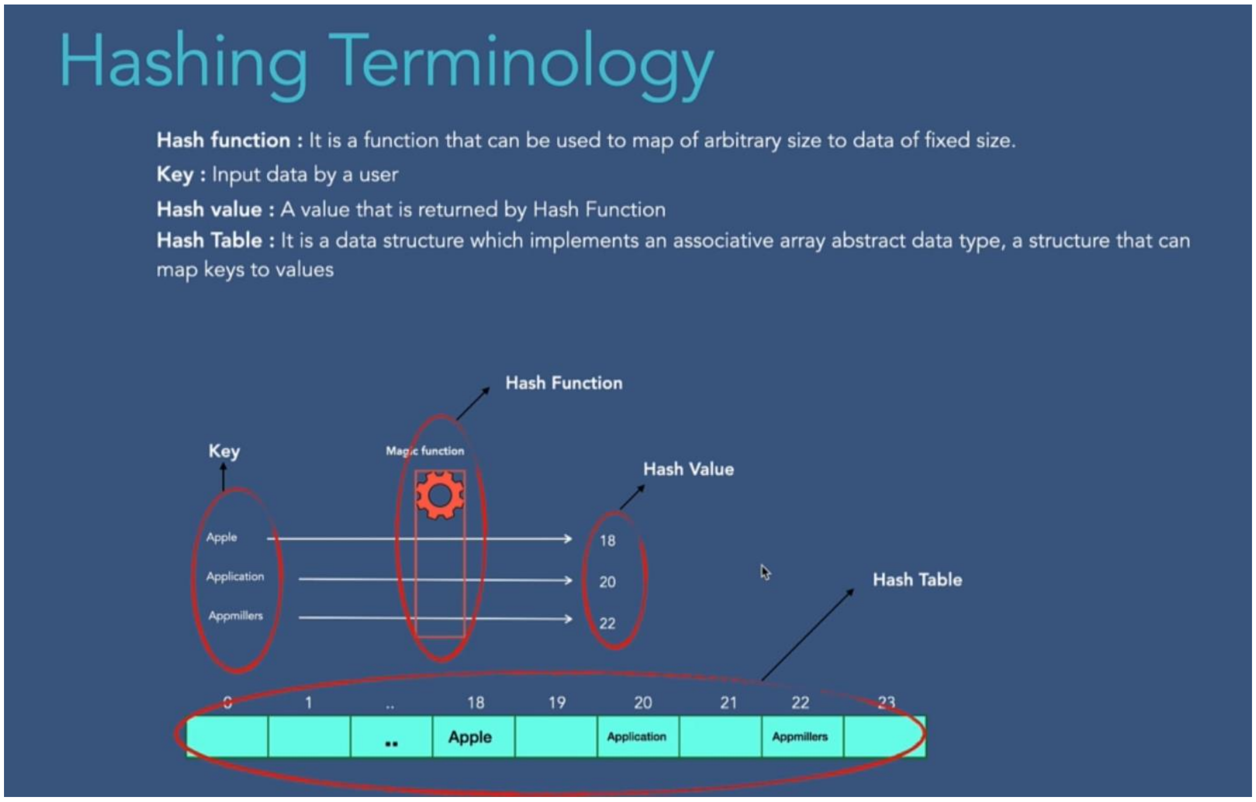
Hashing Terminology

Hash function : It is a function that can be used to map of arbitrary size to data of fixed size.

Key : Input data by a user

Hash value : A value that is returned by Hash Function





Hashing Terminology

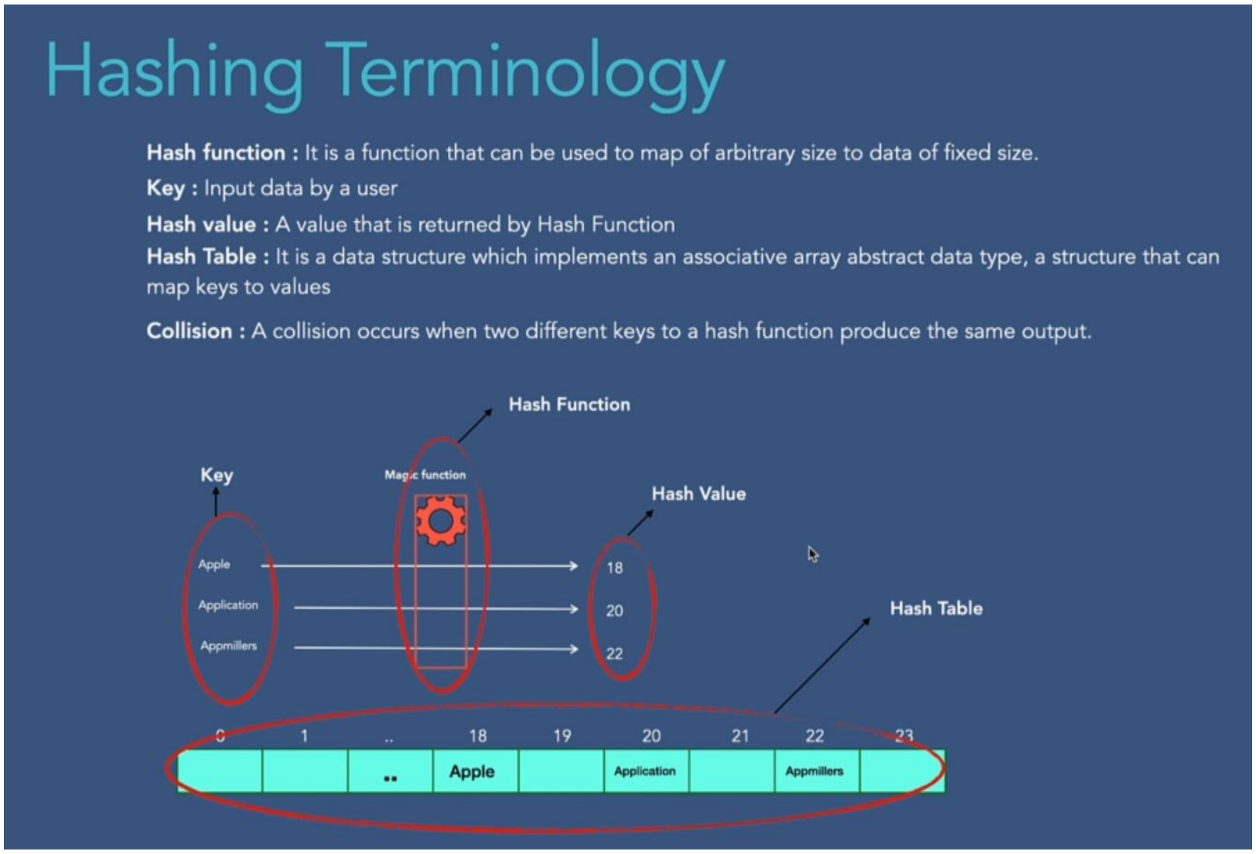
Hash function : It is a function that can be used to map of arbitrary size to data of fixed size.

Key : Input data by a user

Hash value : A value that is returned by Hash Function

Hash Table : It is a data structure which implements an associative array abstract data type, a structure that can map keys to values

Here, we have used Arrays as the Hash Table. Generally, we use Hash Map as the Hash Table.



Hashing Terminology

Hash function : It is a function that can be used to map of arbitrary size to data of fixed size.

Key : Input data by a user

Hash value : A value that is returned by Hash Function

Hash Table : It is a data structure which implements an associative array abstract data type, a structure that can map keys to values

Collision : A collision occurs when two different keys to a hash function produce the same output.

Hashing Terminology

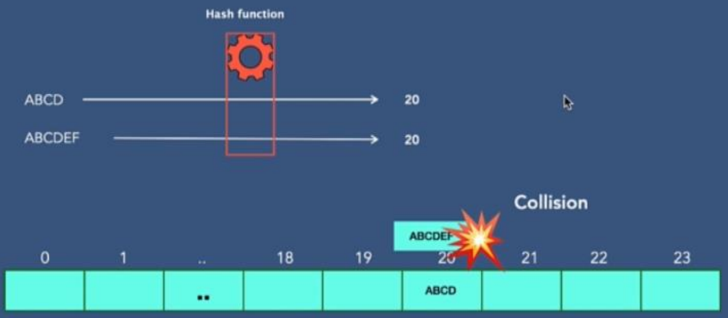
Hash function : It is a function that can be used to map of arbitrary size to data of fixed size.

Key : Input data by a user

Hash value : A value that is returned by Hash Function

Hash Table : It is a data structure which implements an associative array abstract data type, a structure that can map keys to values

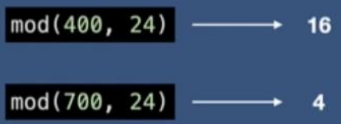
Collision : A collision occurs when two different keys to a hash function produce the same output.

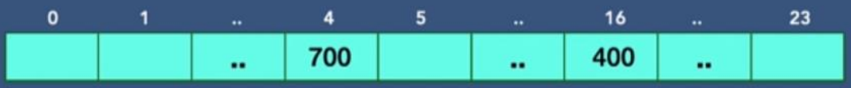


Hash Functions

Mod function

```
int mod(int number, int cellNumber) {
   return number % cellNumber;
}
```





Hash Functions

ASCII function

```
public int modASCII(String word, int cellNumber) {
    int total = 0;
    for (int i=0; i<word.length(); i++) {
        total += word.charAt(i);
        System.out.println(total);
    }
    return total % cellNumber;
}
```

`modASCII("ABC", 24)` $\longrightarrow$ 6

A $\longrightarrow$ 65

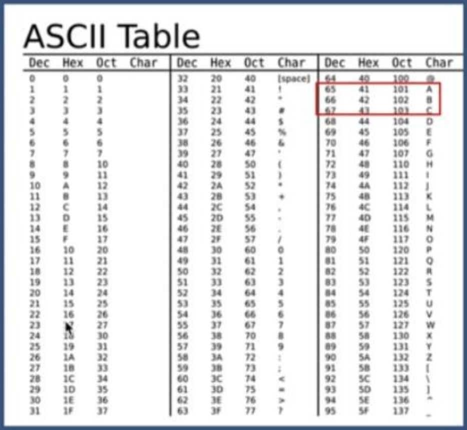
B $\longrightarrow$ 66

C $\longrightarrow$ 67

65+66+67 = 198 | 24

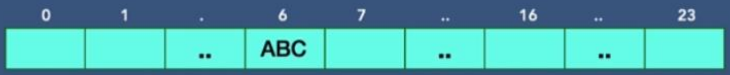
192 | 8

6



ASCII Table

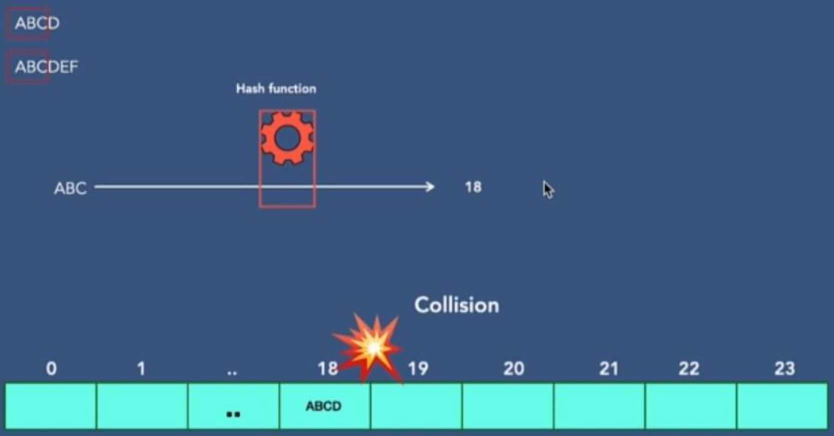
Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@
1	1	1		33	21	41	!	65	41	101	A
2	2	2		34	22	42	"	66	42	102	B
3	3	3		35	23	43	#	67	43	103	C
4	4	4		36	24	44	$	68	44	104	D
5	5	5		37	25	45	%	69	45	105	E
6	6	6		38	26	46	&	70	46	106	F
7	7	7		39	27	47	'	71	47	107	G
8	8	10		40	28	50	(	72	48	110	H
9	9	11		41	29	51	)	73	49	111	I
10	A	12		42	2A	52	*	74	4A	112	J
11	B	13		43	2B	53	+	75	4B	113	K
12	C	14		44	2C	54	,	76	4C	114	L
13	D	15		45	2D	55	-	77	4D	115	M
14	E	16		46	2E	56	.	78	4E	116	N
15	F	17		47	2F	57	/	79	4F	117	O
16	10	20		48	30	60	0	80	50	120	P
17	11	21		49	31	61	1	81	51	121	Q
18	12	22		50	32	62	2	82	52	122	R
19	13	23		51	33	63	3	83	53	123	S
20	14	24		52	34	64	4	84	54	124	T
21	15	25		53	35	65	5	85	55	125	U
22	16	26		54	36	66	6	86	56	126	V
23	17	27		55	37	67	7	87	57	127	W
24	18	30		56	38	70	8	88	58	130	X
25	19	31		57	39	71	9	89	59	131	Y
26	1A	32		58	3A	72	:	90	5A	132	Z
27	1B	33		59	3B	73	;	91	5B	133	[
28	1C	34		60	3C	74	<	92	5C	134	\
29	1D	35		61	3D	75	=	93	5D	135	]
30	1E	36		62	3E	76	>	94	5E	136	^
31	1F	37		63	3F	77	?	95	5F	137	_

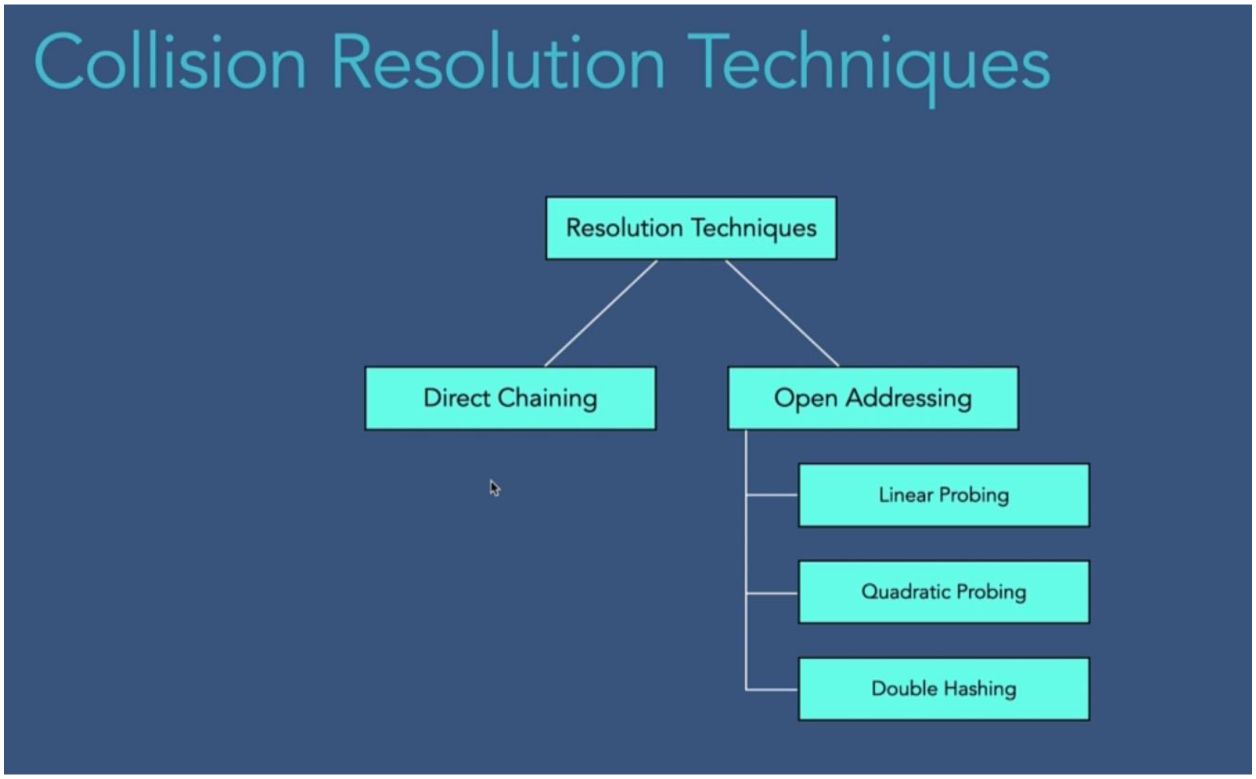


Hash Functions

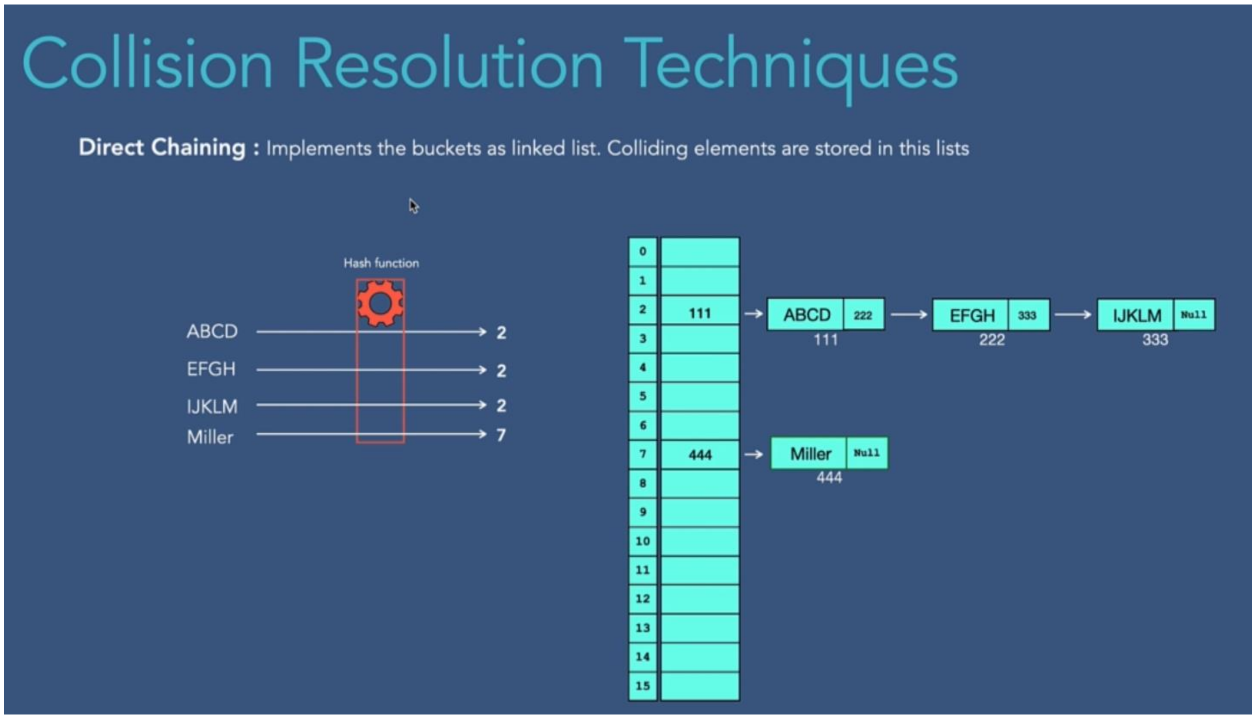
Properties of good Hash function

- It distributes hash values uniformly across hash tables
- It has to use all the input data





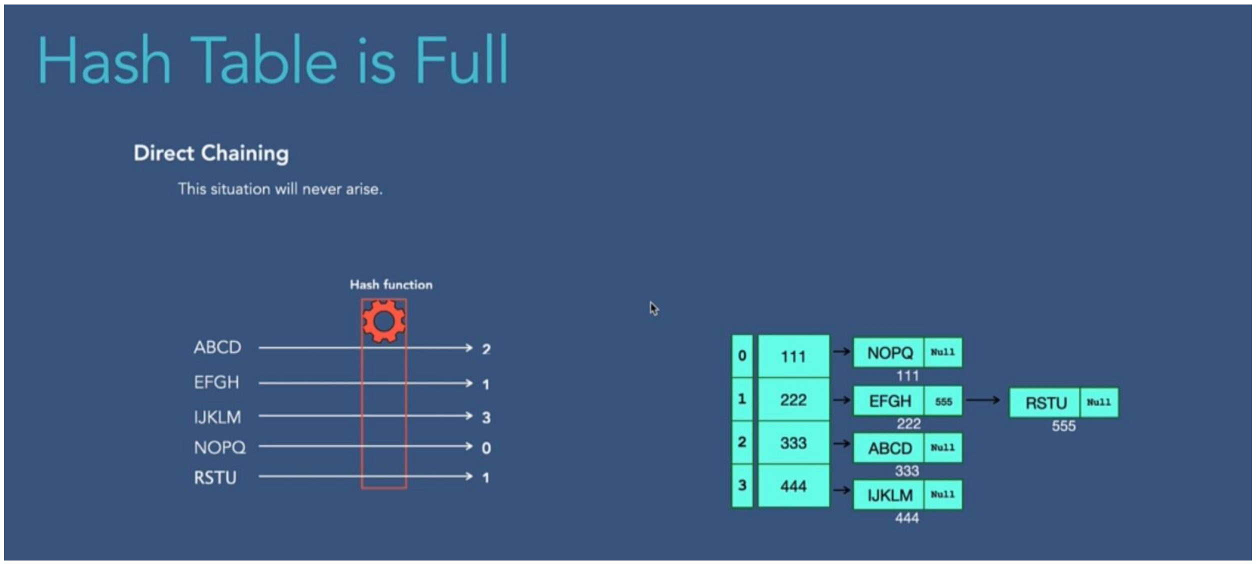
Collision Resolution Techniques



Collision Resolution Techniques

Direct Chaining : Implements the buckets as linked list. Colliding elements are stored in this lists

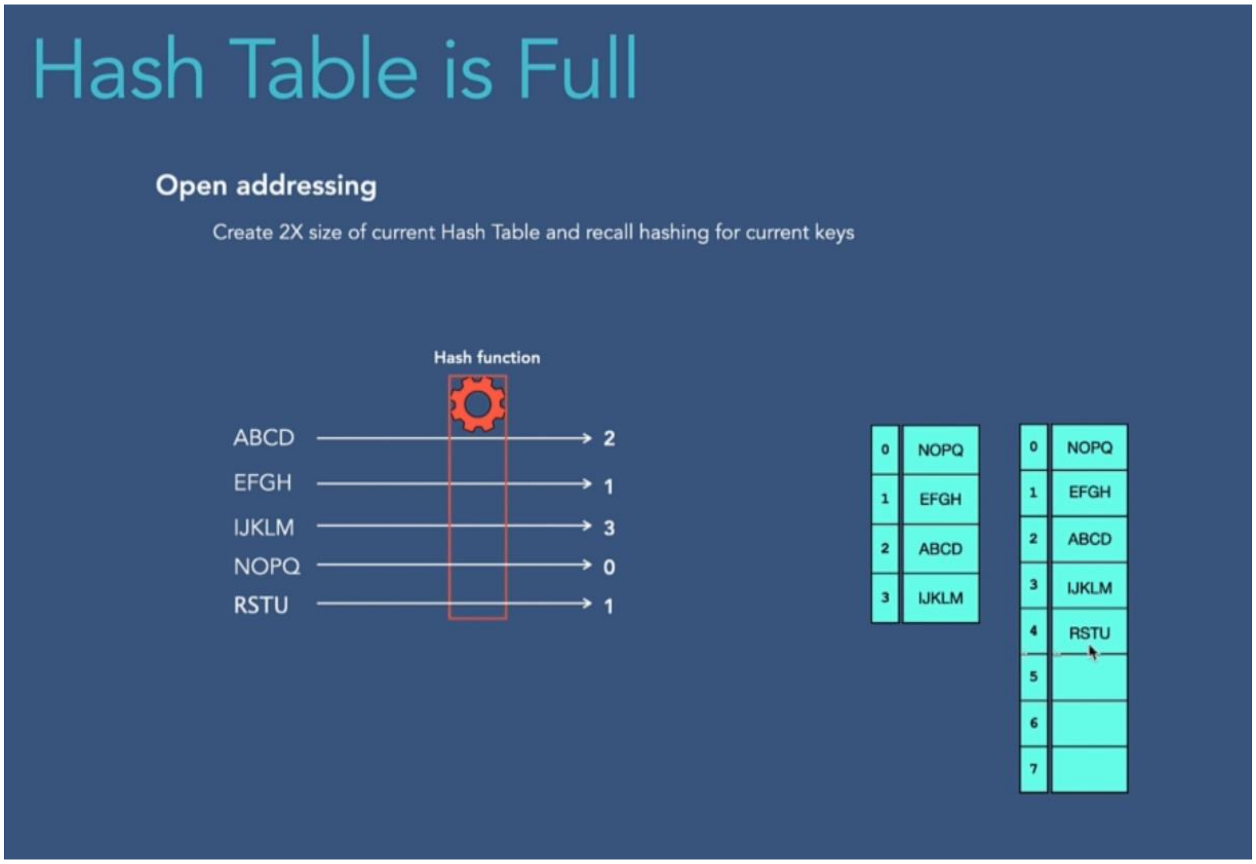
The Hash Table is an array here and Double chaining is implemented with the help of Linked List



Hash Table is Full

Direct Chaining

This situation will never arise.

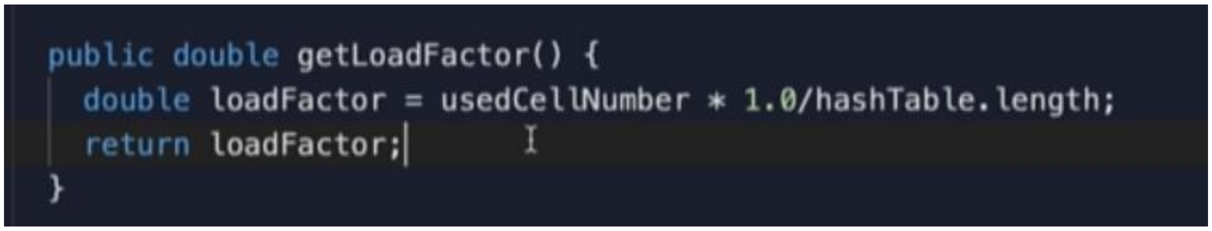


Hash Table is Full

Open addressing

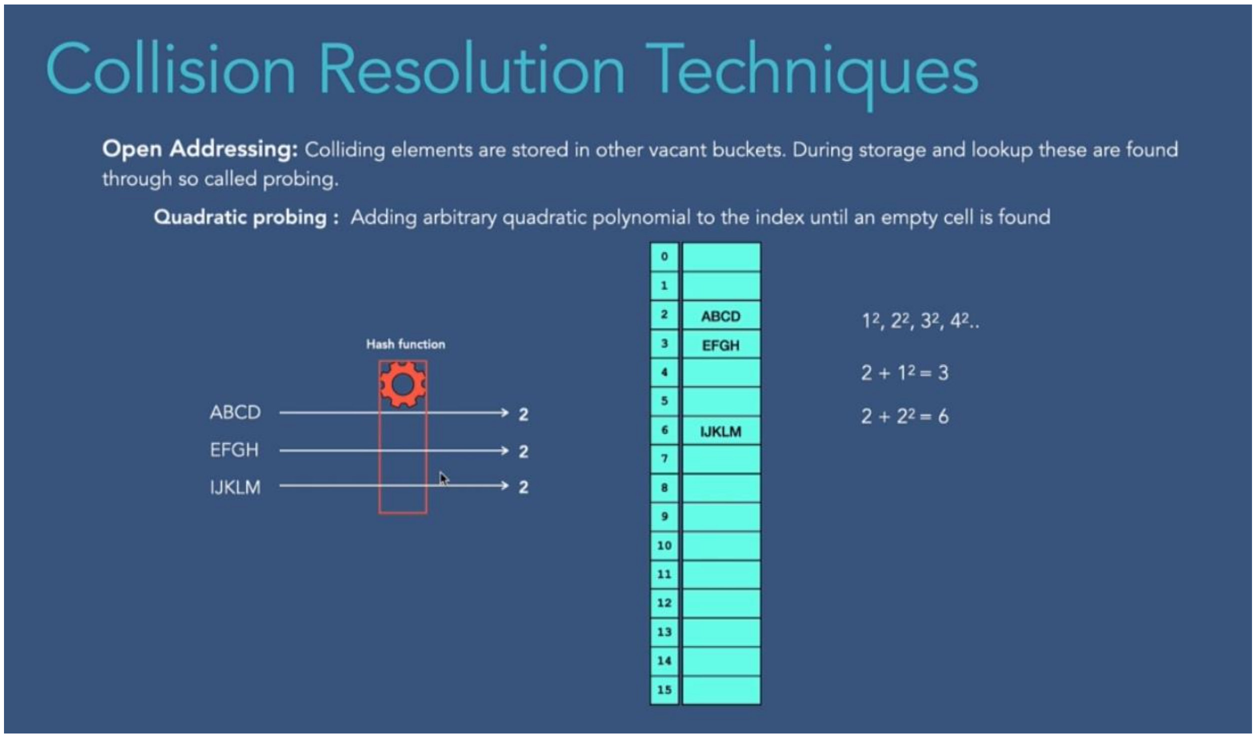
Create 2X size of current Hash Table and recall hashing for current keys

In Open Addressing, if the Hash Table is full, then we increase the size of the Hash Table by Twice and then recall hashing for current keys. Here, for RSTU we have got the same key 1 again which is already filled so it get placed in the next free cells in the Hash Table.



```
public double getLoadFactor() {
  double loadFactor = usedCellNumber * 1.0/hashTable.length;
  return loadFactor;
}
```

It is necessary to create a new Hash Table when the old Hash Table is full. We would find whether the Hash Table is full or not with the help of the Load Factor and then do rehashing, then insert the values to the Hash Table.

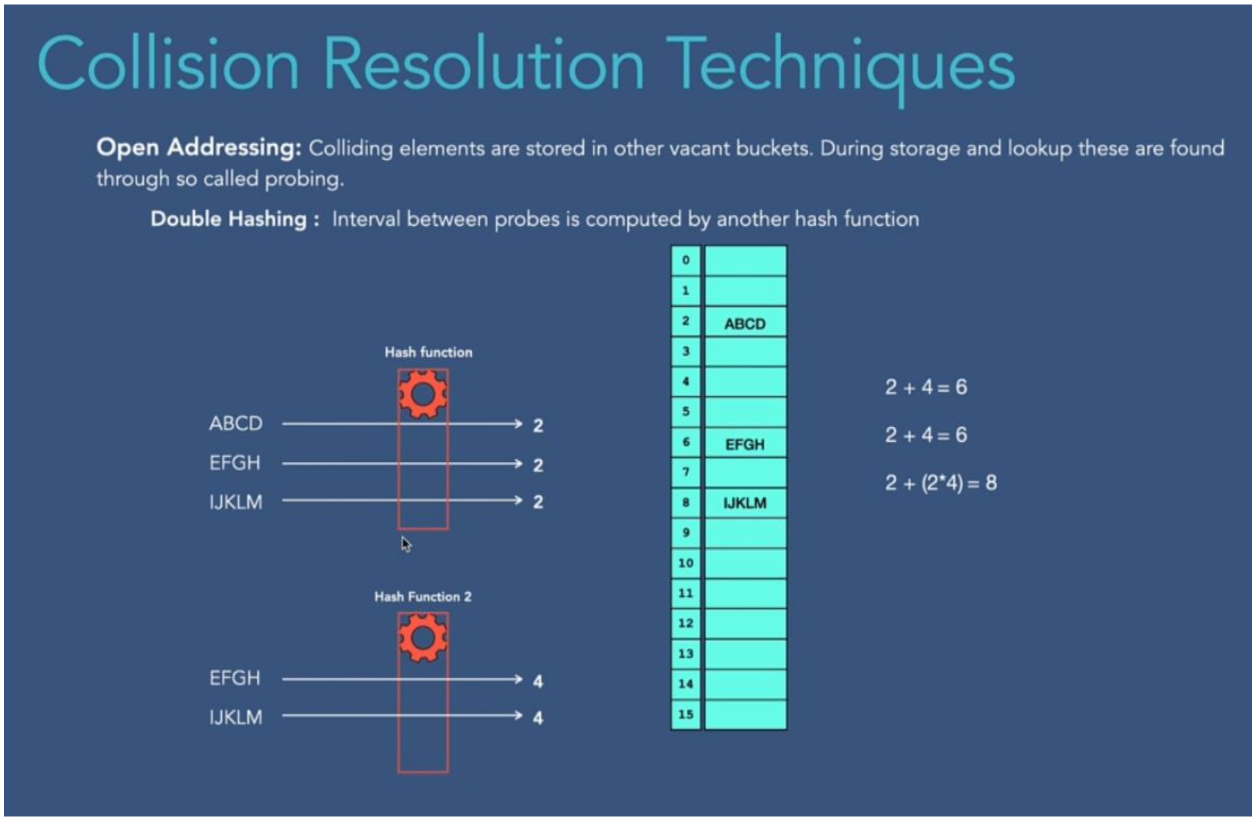


Collision Resolution Techniques

Open Addressing: Colliding elements are stored in other vacant buckets. During storage and lookup these are found through so called probing.

Quadratic probing : Adding arbitrary quadratic polynomial to the index until an empty cell is found

In Quadratic probing, when we find same key for insertion into the Hash Table, we then add polynomials like (1^2,2^2,3^2,4^2…., etc) up until an empty cell is found in the Hash Table. Refer the above image for better understanding.



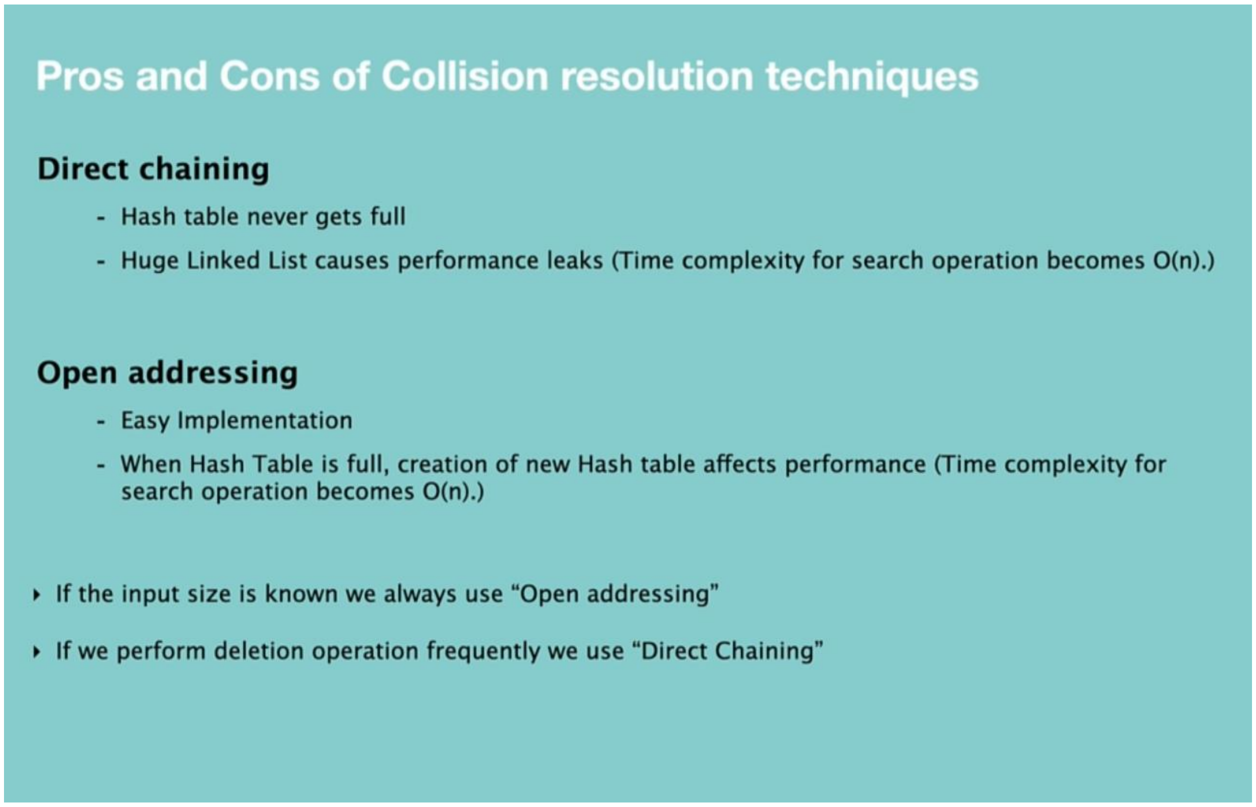
Collision Resolution Techniques

Open Addressing: Colliding elements are stored in other vacant buckets. During storage and lookup these are found through so called probing.

Double Hashing : Interval between probes is computed by another hash function

Here in Double hashing, we have two hash functions. The string gets into the hash function get the key value, if the key value is free in the Hash Table, then we insert the value up there or we have to pass the result of the Hash Function 1 to the Hash Function 2. Even the result of the Hash Function 2 sis filled, the we have to multiply as follows:

2+ (Hash function 1 result * Hash Function 2 result) to get the key index value. Even if this key index value if filled up in the Hash Table then, we have to do the mathematical calculation with higher values. Refer the image above for the better understanding of the concept.



Pros and Cons of Collision resolution techniques

Direct chaining

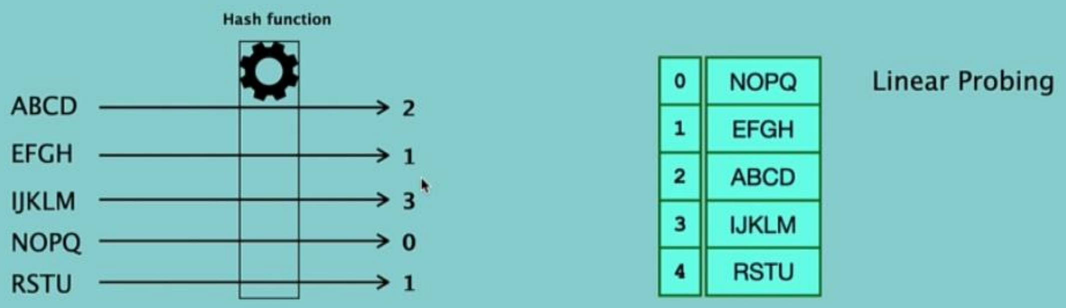
- Hash table never gets full
- Huge Linked List causes performance leaks (Time complexity for search operation becomes O(n).)

Open addressing

- Easy Implementation
- When Hash Table is full, creation of new Hash table affects performance (Time complexity for search operation becomes O(n).)

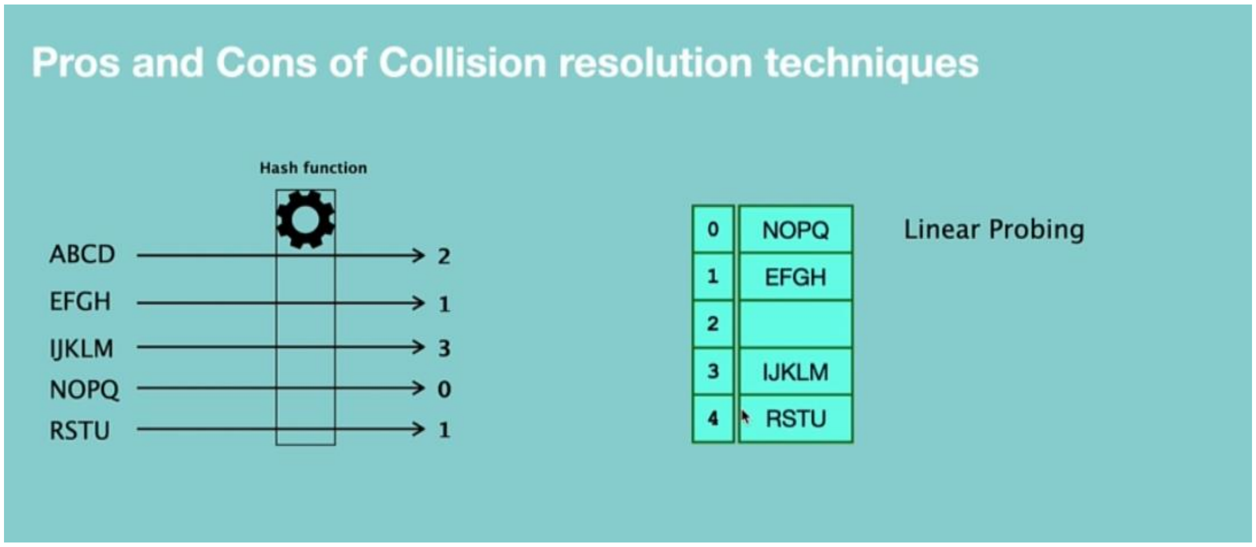
- If the input size is known we always use “Open addressing”
- If we perform deletion operation frequently we use “Direct Chaining”

Pros and Cons of Collision resolution techniques

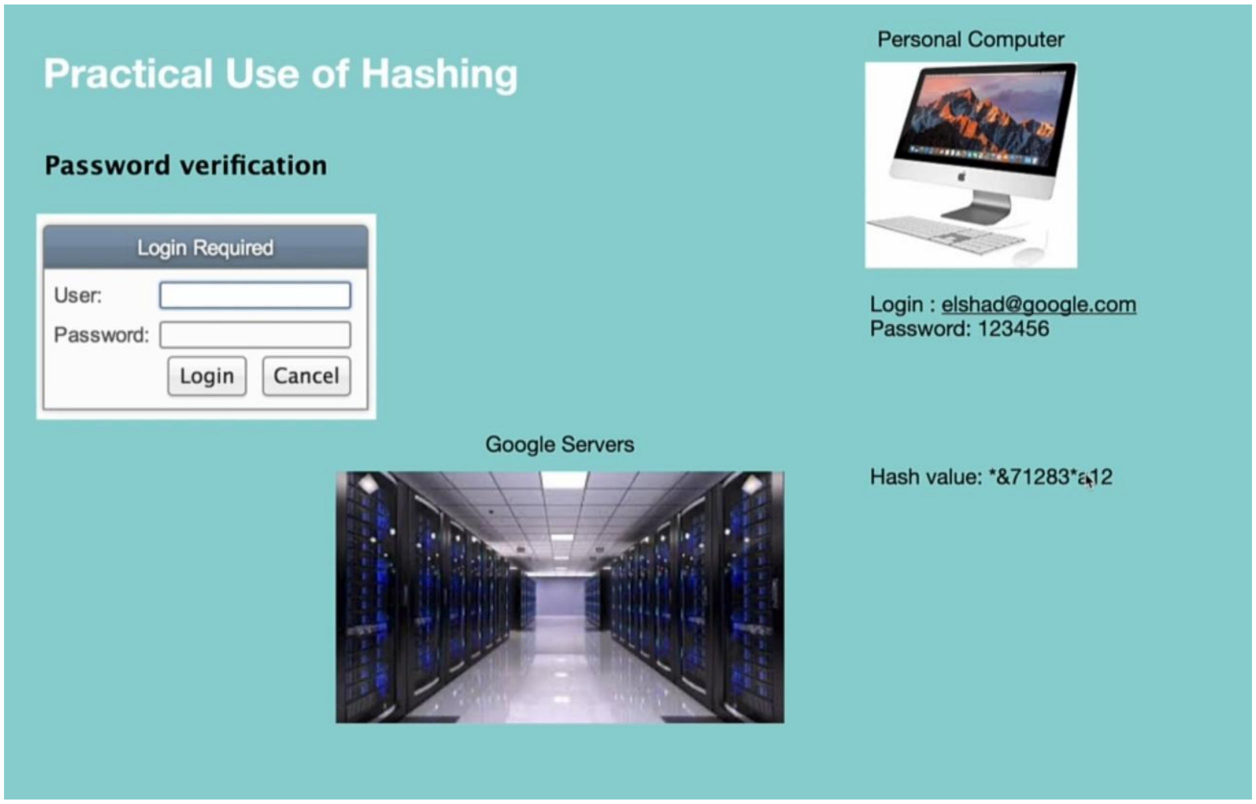


CONS:

Consider the Hash Table above. Suppose if we delete String “ABCD” the Hash Table will get changed as below:



Now for an example, the String "ABCD" which is in the Index value of 2 is deleted from the Hash Table. And we have "EFGH" and "RSTU" with the Hash Values so they get inserted at 1 and 4 position respectively. So, while inserting values into the Hash Table, we would have inserted to the next free cells as the current cell is filled up in the Hash Table by the concept of Linear Probing (Open Addressing) in Hashing. Suppose if we use this function for search operation and search for the String "RSTU" so the hash function returns us with the index value of 1. We then search for the index value of 1 to be equal to "RSTU". It is not equal to "RSTU" so by the concept of Linear Probing (Open Addressing) in Hashing we check for the next cell which is equal to "RSTU" but actually String "RSTU" is present in the index value of 4 in the Hash Table. So, it returns us with String "RSTU" is not in the Hash Table when it is actually present. So, there is inconsistency. So, we should not use the Hashing technique when our application has more delete operation. We can also re-structuring the value which is time consuming. Our ultimate objective is to reduce the time for which we have chosen the Hashing technique. That is the disadvantage of using Hashing. In case of Direct Chaining, the delete operation is very simple so it does not cause the empty cells in Hash Table.



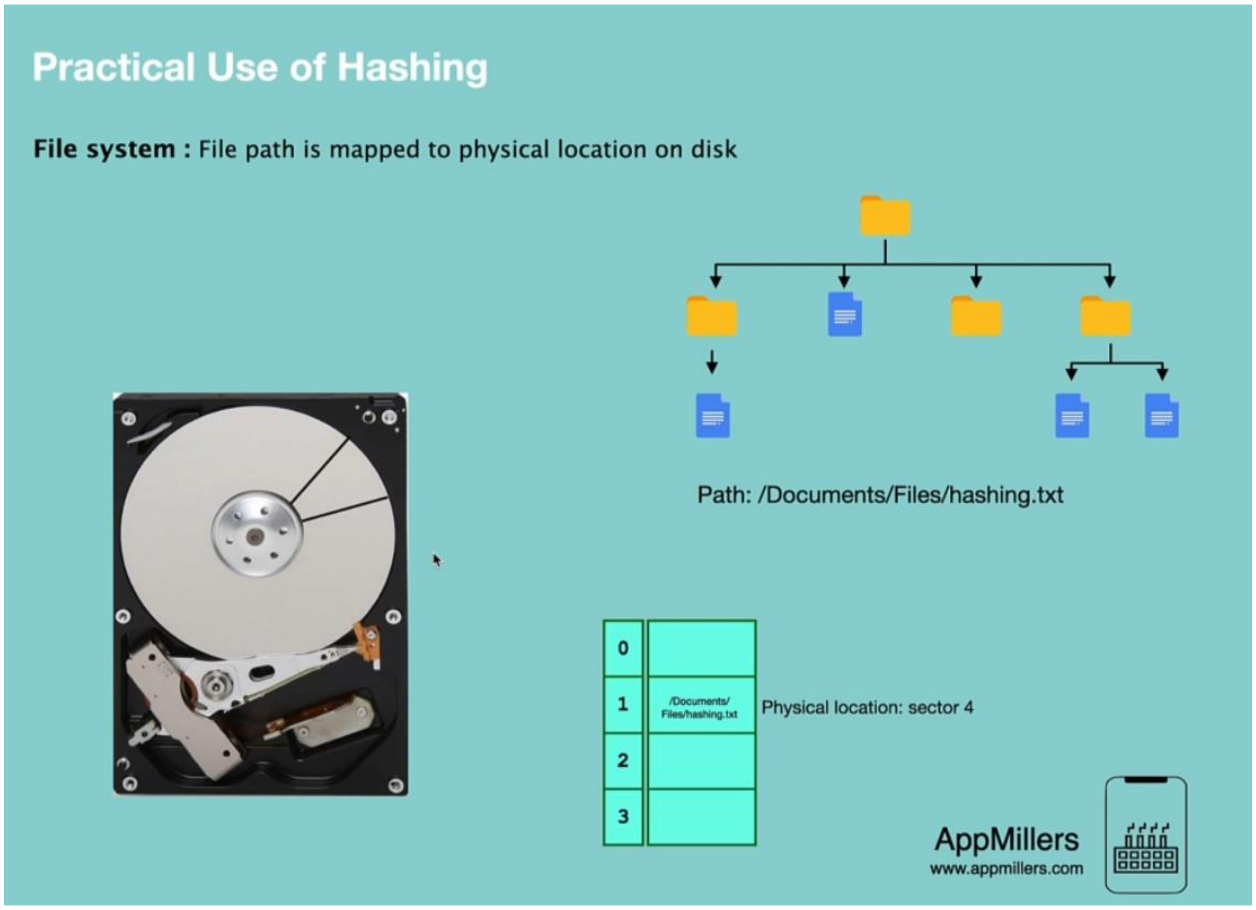
Practical Use of Hashing

Password verification

Login : elshad@google.com
Password: 123456

Hash value: *&71283*a12

Hashing can be used in password encryption and managing the user credentials. For an example we create an account and the Google servers store our credentials after Hashing to protect our passwords from Hackers. When trying to login again, the password entered by the user gets into the Hashing Function and the result hashed value is checked with the hashes present in the Google servers and if similar the user gets authenticated. In this way, Hashing is used in real world.

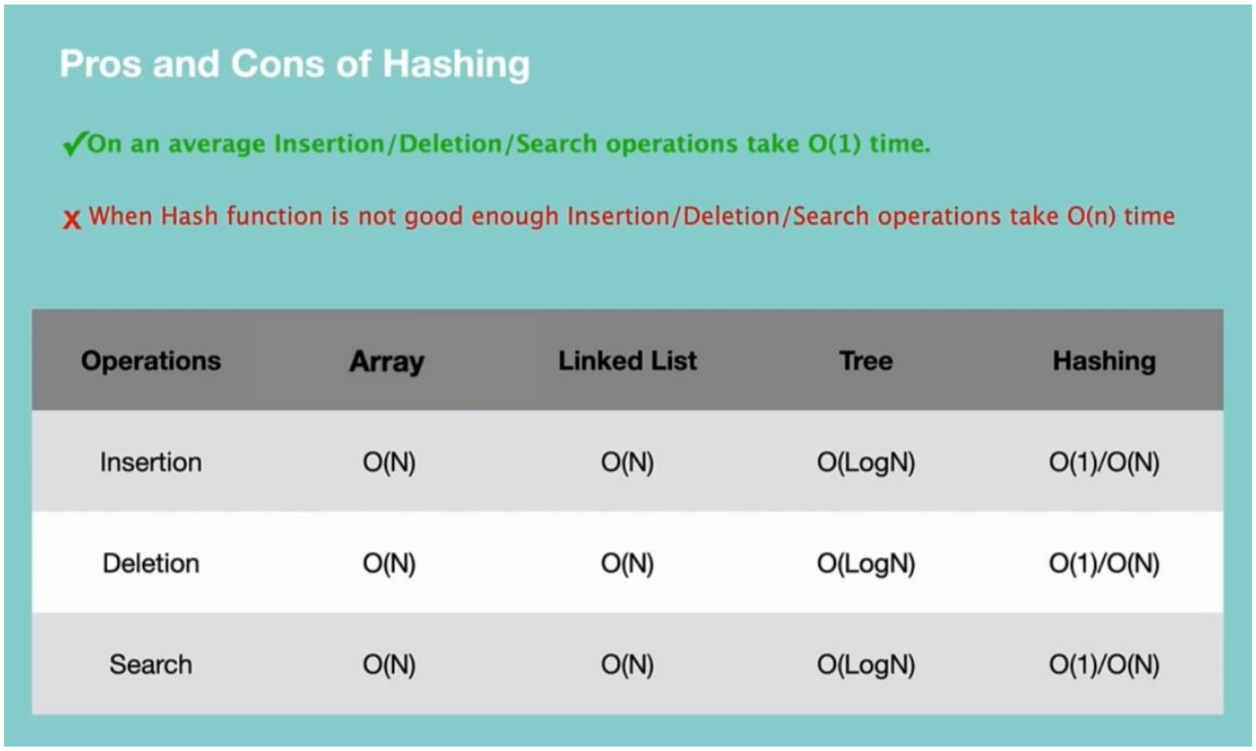


Practical Use of Hashing

File system : File path is mapped to physical location on disk

Path: /Documents/Files/hashing.txt
0
1
/Documents/Files/hashing.txt
2
3
Physical location: sector 4
AppMillers
www.appmillers.com

Just relate with the above example in the similar way and map that to this example.



Pros and Cons of Hashing

✓On an average Insertion/Deletion/Search operations take O(1) time.

x When Hash function is not good enough Insertion/Deletion/Search operations take O(n) time

Operations	Array	Linked List	Tree	Hashing
Insertion	O(N)	O(N)	O(LogN)	O(1)/O(N)
Deletion	O(N)	O(N)	O(LogN)	O(1)/O(N)
Search	O(N)	O(N)	O(LogN)	O(1)/O(N)