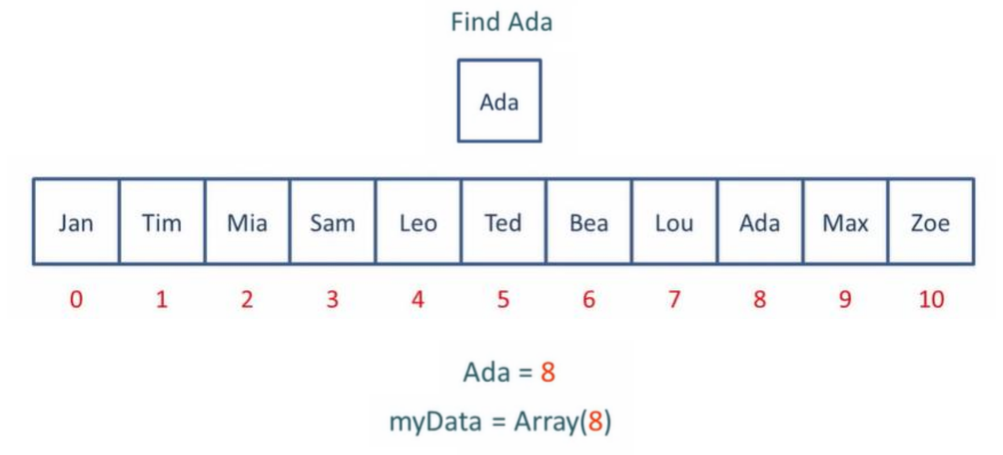
Hash Table & Its Operations

Hash table is a data structure that allows very fast retrieval of data, no matter how much data there is.
Application of Hash Table-

- *Database indexing*
- *Caching*
- *Program Compilation*
- *Password Authentication*
- *Error Checking etc.*

Why Hash Table?

Problem: Find Ada?
Solution 1: Use linear search (Brute force approach).
Drawback: Slow approach if array is of big size since need to check each value one by one.



Solution 2: What if we know the index? We can directly locate the value instead of searching sequentially, irrespective of how much big array or which position its located

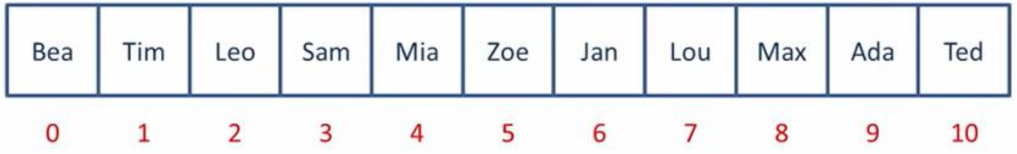
Solution 2, But How?

Solution 2: What if we know the index? We can directly locate the value instead of searching sequentially, irrespective of how much big array or which position its located

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

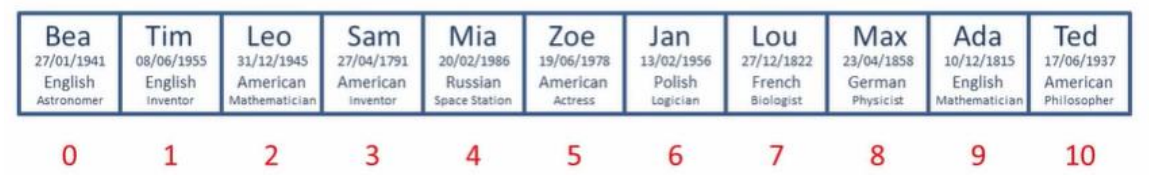
Index number = *sum ASCII codes* Mod *size of array*

Find Ada Ada =
(65 + 100 + 97) = 262
Find Ada 262 Mod 11 = 9
myData = Array(9)



Hash Table Structure

Rather than just storing and individual item data, hash tables are often used to store Key, Value pairs.
That time Hash Table are also called as Hash Map
e.g. Key-> Name and Value-> Date of birth



Bea	Tim	Leo	Sam	Mia	Zoe	Jan	Lou	Max	Ada	Ted
27/01/1941	08/06/1955	31/12/1945	27/04/1791	20/02/1986	19/06/1978	13/02/1956	27/12/1822	23/04/1858	10/12/1815	17/06/1937
English	English	American	American	Russian	American	Polish	French	German	English	American
Astronomer	Inventor	Mathematician	Inventor	Space Station	Actress	Logician	Biologist	Physicist	Mathematician	Philosopher
0	1	2	3	4	5	6	7	8	9	10

Key → Name
Value → Object

Hashing Algorithm (Hashing Function)

Hashing Algorithm is the calculation applied to a key (which may be a very large number or very large string), to transform it to relatively small index number that correspond to the position in a hash table.
This index number is effectively a memory address.
The beauty of such hash algorithm is that if you feed the same input into it again and again, it will give the same index each time.
If Key is numeric value - Divide the key by number of available addresses, n, and take the remainder.

address=key Mod n

If Key is alphanumeric value - Divide the sum of ASCII codes in a key by the number of available addresses, n, and take the remainder.

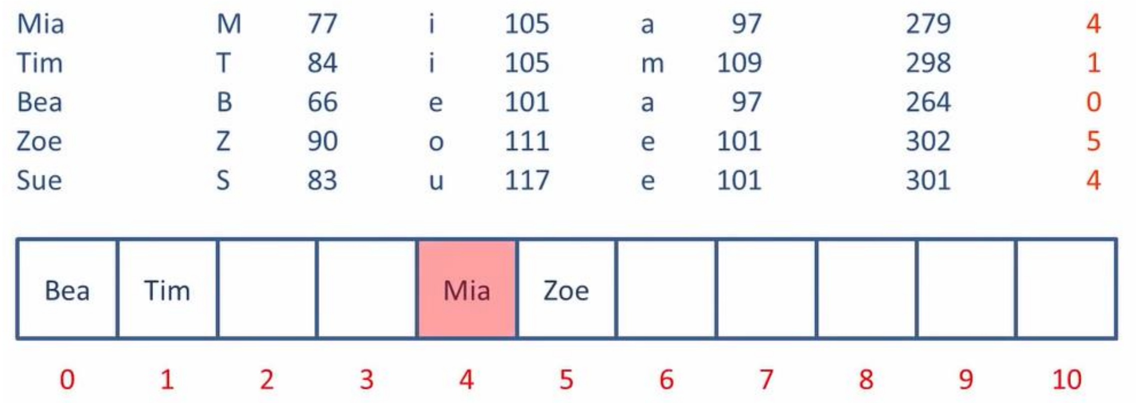
*Another method which can be applied is called **folding method** divides key into equal parts then adds the parts together*
e.g.

Telephone no: 01452 8345654 becomes 01+45+28+34+56+54=218
If the size of hash table is n then divide by n and take the remainder.

There are lots of different Hashing Algorithms to choose from, some more appropriate than other depending upon nature of your data.

Collisions

Hashing Algorithm might generate same index for two data elements which results into situation called 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

Solution:

1. Open Addressing Techniques
   - Linear Probing
   - Double Hashing
2. Closed Addressing Techniques
   - Chaining

Open Addressing Techniques – Linear Probing

Clash at position 4 we check for next available position for Sue since Zoe took it Sue need to be placed inside 6th position.
For Rae we did not find ay position till 7th index hence it needs to be placed in 8th index.

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

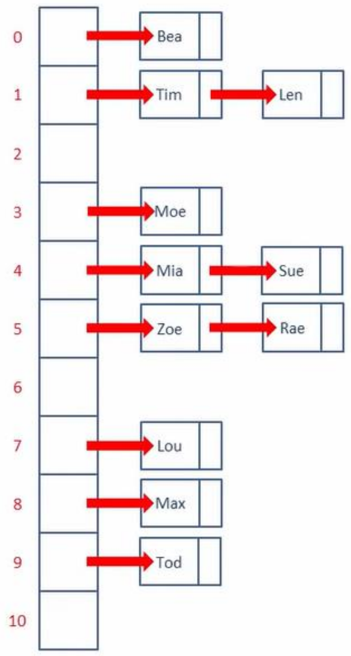
Bea	Tim	Len	Moe	Mia	Zoe	Sue	Lou	Rae	Max	Tod
0	1	2	3	4	5	6	7	8	9	10

Open Addressing Techniques – Linear Probing

- *If calculated address is occupied, then the linear search is used to find the next available slot.*
- *If linear probing gets till end of array and it still cannot find the free space it might cycle around at the beginning of the array & continue searching from there.*
- *Now when you want to search any item in hash table the hashing function is applied again & if there are collisions, and some items are not at their calculated positions then finding the item will also involve linear probing i.e a linear search.*
- *The more items are there in hash table the more likely you are to get collisions when you insert even more data.*
- *One way to deal with this is to make hash table bigger than needed for the total amount of data you will be expecting, perhaps such that only 70% of the hash table is ever occupied.*
- *The ratio between the number of items stored and the size of the data array is known as the* ***Load Factor.***
- *If the hash table is implemented as a resizable dynamic data structure it could be made to increase in size automatically when the load factor reaches a certain threshold**.***
- *In an ideal world every item will be stored in the hash table according to its calculated index in this best-case scenario the time taken to find any particular item is always the same but you can imagine the worst case scenario depending upon the nature of the data used to calculate the index values and depending upon the appropriateness of the hash algorithm some items may require a linear search of the whole table in order to be found.*
- *As long as the load factor is reasonably low open addressing with linear probing should work reasonably well.*

Chaining

For Collision values separate linked list is maintain and on searching the same is iterated till the item is located.
Remember the lookup is quicker here but traversing Linked list also comes with cost.
If the load factor is low better choose open addressing



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

Find Rae
280 Mod 11 = 5
myData = Array(5)

$\boxed{\text{Rae}}$

Objective of Hash Function

If you know all of the keys in advanced, then its theoretically possible to come up with perfect hash function.
One that will produce unique index for each and every data item.
In fact if you know the data in advanced you can come up with perfect hash function that uses all of the available spaces in the array.
Objective of hash function-

1. *It should minimize the collisions*
2. *Uniform distribution of hash values*
3. *Easy to calculate*
4. *Should include suitable method to resolve any collisions that do occur.*

Hash Tables are used to index large amount of data
Address of each key is calculated using the key itself.
Collisions are resolved either with Open or closed addressing.