(5) Hash Table :

If we have a dataset in which we have pairs (key & value)

For example:

Stock Price data with 'date' and value at that particular date.

→ So if we want to get a value by giving a particular date, how we can do it ?

↳ So first way is we write code where we split data (store it in list).

↳ And then get data by iterating & match by value thing.

↳ So complexity will be O(n) to find any matching value.

↳ Above we used Array (list in python)

→ Now if we simply make a python dictionary and store the data in key-value pair - Like

```
stock_prices = { }
with open("stock.csv", "r") as f:
    for lines in f:
        token = line.split(',')
        day = token[0]
        price = token[1]
        stock_prices[day] = price
```

↳ Assign days the value of prices in dictionary

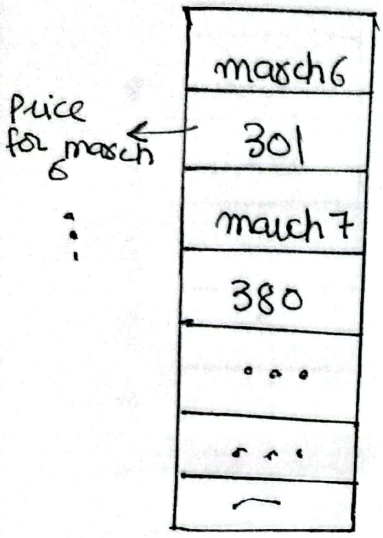
So using this program now we can access values like :

```
stock_price['any day']
```

↳ Which give O(1) complexity

⇒ Underlined data structure of dictionary is hash table.

→ How our stock_prices were stored in list:



⇒ So everything was stored in contigious memory locations.

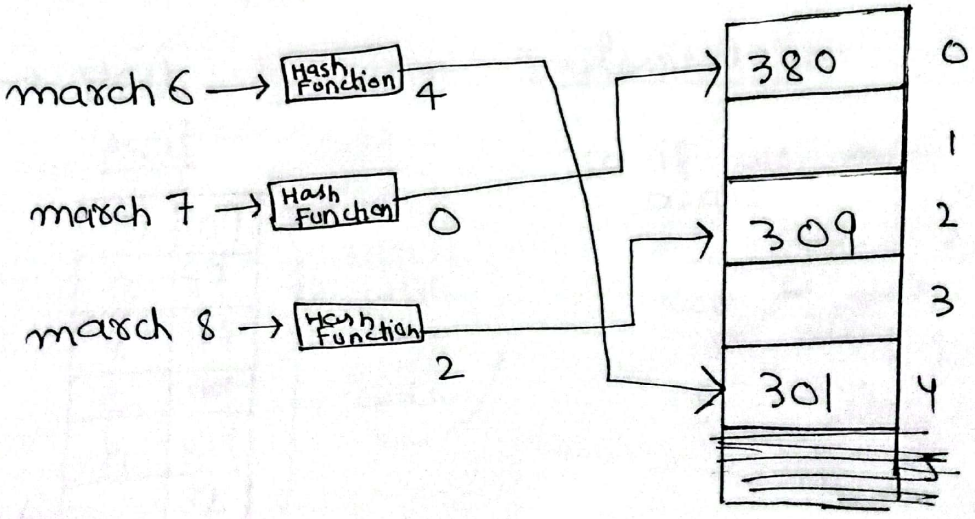
↳ As the list usually stores.

→ How our stock_prices were stored in dictionary:

① Firstly it will allocate a list of some size.

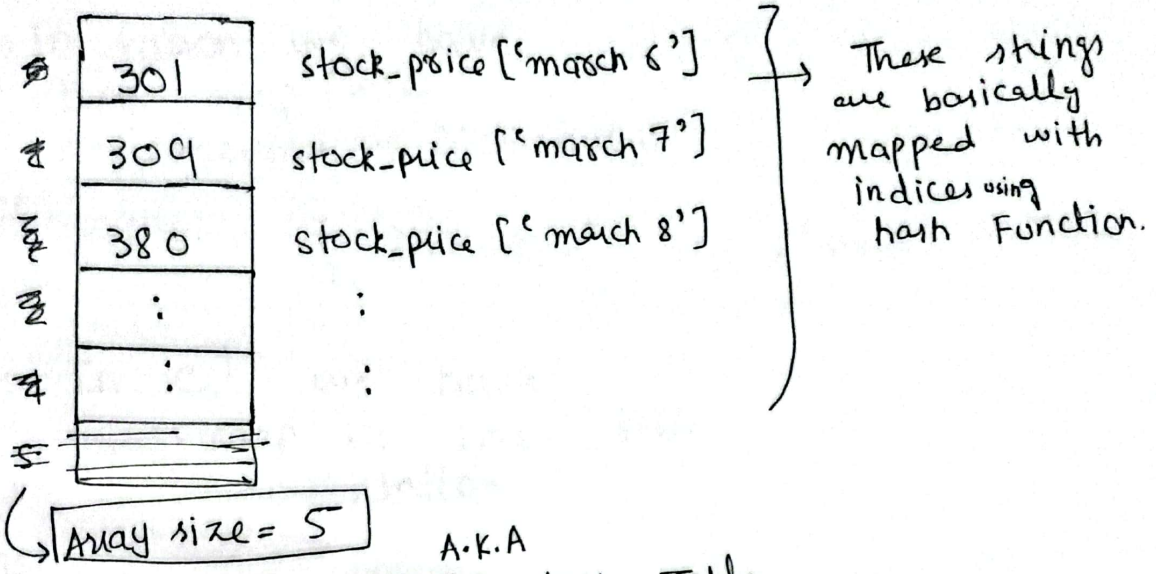
② And it will map string key to its value stored in list.

↳ It will get the index value for mapping using hash function.



→ All the thing hash function is doing is finding index of element and mapping it with key.

↳ In hash map you have a string index.



↳ Array size = 5

→ Hash map A.K.A ⟺ Hash Table

Hash Function:

↳ It gives us a integer index from a string.

'march 6' → Hash Function → 6

↳ There are alot of methods to implement hash function.

→ Hash Function Implementation using Ascii Code:

m	109
a	97
r	114
c	99
h	104
space	32
6	54

→ letters with their ascii codes.

so if we sum these ascii

=> SUM = 609

Now to find index:

As array size = 5

MOD (609, 5) = 4

(% ↑ MOD, 609 ↑ sum, 5 ↑ size of array, 4 ↑ index)

→ In hash maps look by key is O(1) on average.

→ Insertion/Deletion → O(1) on average

↳ Order of 1
Big O complexity

↳ In Python we have hash map / table implemented as Dictionary:

Syntax

```
prices = { 'march 6' : 301 , 'march 7' : 302 }
```

↳ In c++ we have std:map as hash table implementation.

syntax

```
std::map <string , int > prices;
prices ['march 6'] = 310;
prices ['march 7'] = 302;
```

⟹ Go check Implementation code.

⑥ **Collision Handling In Hash Tables :**

What if we have a value for a key in our hash map and then we enter another value for the same key. (How we will store 2 values ?)

↳ we will loose value as it will be overwritten.

↳ For this we can use

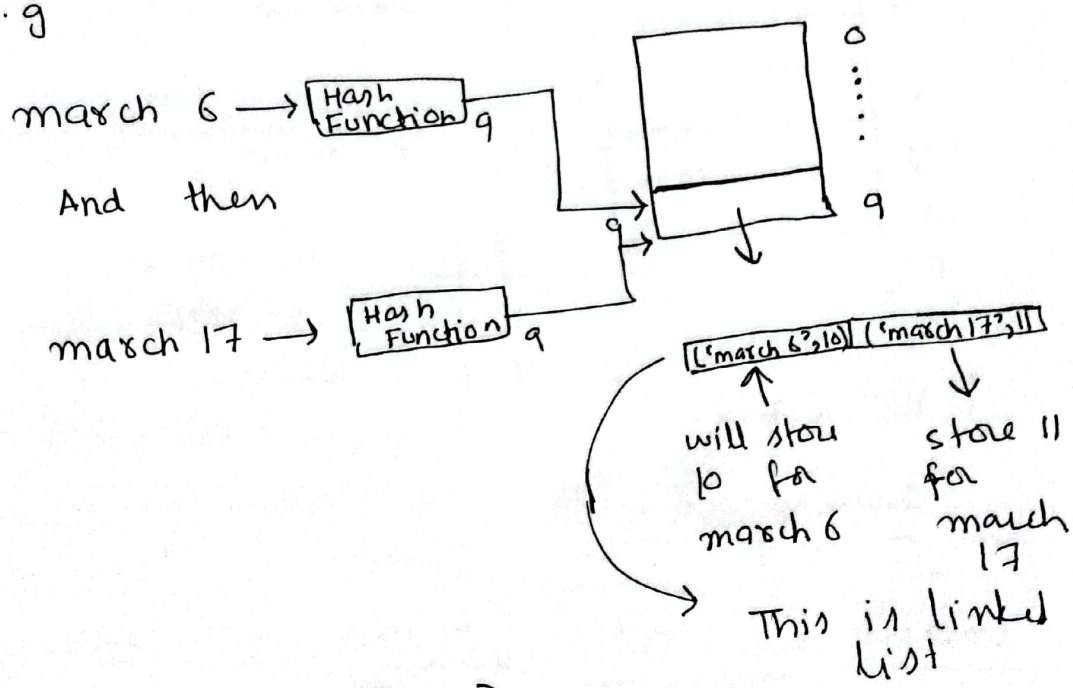
(i) Chaining or Separate Chaining approach -

↳ This is called as collision.

↳ Collision can happen if hash function generate same index for different keys.

⇒ So in chaining we try to store a "linked list" in the place when collision comes up.

e.g



⇒ For this the Time complexity can go to O(n) → Look up complexity

↳ same as search Time complexity of linked list -

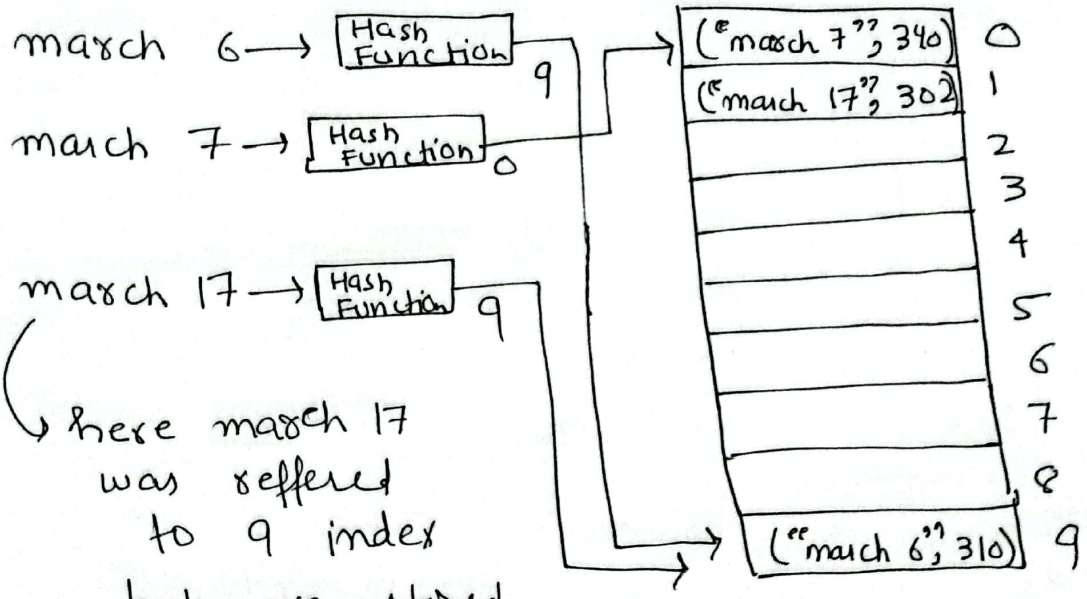
(ii) Second approach to solve this collision problem is:

Linear Probing

↳ In this if collision happens we store the new key & value next to the pre existed key value at that index (on which collide happens).

↳ If next space is filled we will go to next space means we will linearly search for empty space.

e.g



↳ here march 17 was reffered to 9 index but we stored it at index 1 b/c when we seached linearly from 9 to 0 (0 was not empty) so we went to 1 & it was empty so we stored it there.

How to Program a Hash Table?

① We will make a class that will be our hash table.

② We will make a list of specific size (in def __init__( ) method)

③ Method get_hash() to get the index for the element.

④ We will use __getitem__ (self) that is operator used to get value from hash map as we get value from dictionary in python.

↳ e.g print (dict['key'])

↳ (i) In this method we will get the index from get_hash().

(ii) Then simply return that indexed value. (whatever is at that index)

⑤ We will use __setitem__ (self) that is operator used to set / add new (~~key~~ value) ~~pair~~ in our hash map. (In python style)

↳ e.g dict['key'] = value

↳ (i) we will pass key to get_has() and it will give us index.

(ii) Simply insert that value in that index.

⑥ We will use --delitem-- (self) to delete a specific key's value from hash map. (In Python style)

↳ e.g del dict['key']

↳ (i) Find the index where value is stored.

(ii) Make that index equal to None.

How to apply Chaining on hash map to avoid collision?

Programming Approach

① In this we will make list of elements and every element will be a list as we will store linked list in element if collision happens.

```
self.arr = [[] for i in range(self.size)]
```

② Here again use --getitem-- ().

↳ (i) In this we will get a index of another array. so we need to iterate on this to find our value by comparing key.

↳ (ii) If first thing stored in any array (inner array) matches with our key we will return second word (that will be our key)

③ we will use __setitem__() to store new (key, value) in our hash map. Here we will insert tupple.
↳ (key, value)

↳ (i) Here we will get key & value as argument.

↳ (ii) We will get the index where we will store that tupple.

↳ (iii) Here we will iterate in that element (array) and firstly we will check if we have that key already placed with any other value. In this we will just modify that value with new one.

↳ If key is new we will simply append the tupple at that array.

④ we will use __delitem__() to delete any specific key, value pair & it (pair) can be in the linked list.

↳ (i) we will iterate at the array after getting its index.

↳ (ii) Then we will check every array's first word with our if matches

```
del self.arr[h][index]
```

gives index of array

index where that key is stored

HOW to Approach Collision's solution with Linear Probing :

① We have a method called get-prob-range() that takes index & will return list of index after it + starting index till that index.

↳ e.g If we give 7 as index & we have total = 9

[*range(index, len(array)] + [*range(0, index)]

↓ that we give ↓ returns last index

we will get:

[7, 8, 9, 0, 1, 2, 3, 4, 5, 6]

↳ It's function is to give us indecies linearly so we can check for empty space.

↳ Rest everything explained in code.