HASHING

What is Hashing?

- Hashing is the process of mapping large amount of data item to smaller table with the help of hashing function.
- Hashing is also known as Hashing Algorithm or Message Digest Function.
- It is a technique to convert a range of key values into a range of indexes of an array.
- It is used to facilitate the next level searching method when compared with the linear or binary search.
- Hashing allows to update and retrieve any data entry in a constant time O(1).
- Constant time O(1) means the operation does not depend on the size of the data.
- Hashing is used with a database to enable items to be retrieved more quickly.
- It is used in the encryption and decryption of digital signatures.

What is Hash Function?

- A fixed process converts a key to a hash key is known as a Hash Function.
- This function takes a key and maps it to a value of a certain length which is called a Hash value or Hash.
- Hash value represents the original string of characters, but it is normally smaller than the original.
- It transfers the digital signature and then both hash value and signature are sent to the receiver. Receiver uses the same hash function to generate the hash value and then compares it to that received with the message.
- If the hash values are same, the message is transmitted without errors.

It has a key and a value. So, whenever we use ‘dict’ we can do hashing. Further details will be covered during LINK-List. For now, understand that whenever we use dictionary, it will be hashing.

Solving the above problem using ‘dict’:

If d = {} → the ‘dict’ would be empty. we will check for the first element in ADOBECODEBANC and we can put A was present at the 0^{th} index as A: 0 and the max_length of sub-string would be ‘1’.

Now, at D, as D is not present in the ‘dict’, we can add D and mention its index as 1. (D:1).

The starting point of the sub-string would be at the starting. We can have a variable at start = 0 and end will be at D. The max_length would be (end – start + 1).

At O, we will add it to ‘dict’ as O:2 and will continue;

A: 0, max_length = 0
D: 1, max_length = 1
O: 3, max_length = 2
B: 3, max_length = 3
E: 4, max_length = 4
C: 5, max_length = 5

Now, the end will go to ‘O’. Since O is getting repeated, we have to shrink the window from left to the position of the repeated O.

We will remove A and move the start to D.
We will remove D and move the start to O.
We will remove O and move the start to B.

We will remove all the index from start = 0 to start = the first occurrence of ‘O’. So, the start value will be at **3** and the end would be at 7 – 3 + 1 = 5. It’s less than max length, so we will not update the max_length.

Now, end is at E, and start is at B.

So, we will remove B and start will be at E; we will remove E and start will be at C.

C: 5, max_length = 5
O: 6
D: 7
E: 8
B: 9
A: 10
N: 11

Once we reach C, we will again start shrinking C and start will be at O. For start = O it’s index would be at 6 and the end = 12.

max_length = end – start + 1
= 12 – 6 + 1 = **7**

CODE:

```
def solve(s):
    char_map = dict()
    n = len(s)
    start = 0
    end = 0
    max_length = 0


    while end < n:
        if s[end] not in char_map: # if the end is not in char_map,
            char_map[s[end]] = end # add it to the char_map.
            length = end - start + 1
            max_length = max(max_length, length)
            end += 1
        else:
            while s[end] in char_map:
                char_map.pop(s[start])
                start += 1
            # char_map[s[end]] = end
            # end += 1


    return max_length

if __name__ == "__main__":
    s = "ADOBECODEBANC"
    print(solve(s))
```

Time complexity: O (n)
Space Complexity: O (26)

Q) Given an array A and sum S return **True** if there is any pair in this whose sum is S otherwise return **False**

Ex:

A = [2, 5, 1, 9, 6, 7]

S = 7, **T**rue, (4 + 3)
S = 100, **F**alse
S = 12, **F**alse
S = 6, **T**rue (4 + 2)

1st Approach – Brute-Force Method:

```
for i in range(n):
    for j in range(i + 1, n):
        if A[i] + A[j] == s:
            return True
        return False
```

2nd Approach – Two Pointer Approach:

For A = [1, 2, 3, 4, 6] which is sorted,

Ex: 1 s = 10

left = **1**; right = 6, A**[left]** + A**[right]** = 7 < s (= 10), **left** would be **increased** as the array is sorted in ascending order.
left = **2**; right = 6, A**[left]** + A**[right]** = 8 < s (= 10), **left** would be **increased**
left = **3**; right = 6, A**[left]** + A**[right]** = 9 < s (= 10), **left** would be **increased**
left = **4**; right = 6, **A[left] + A[right] = 10 = s**.

Ex: 2; s = 2.

left = 1, right = **6**, A**[left]** + A**[right]** = 7. Since 7 > s (=2), **right** would be **decreased**.
left = 1, right = **5**, A**[left]** + A**[right]** = 6. Since 7 > s (=2), **right** would be **decreased**.
left = 1, right = **4**, A**[left]** + A**[right]** = 5. Since 7 > s (=2), **right** would be **decreased**.
left = 1, right = **3**, A**[left]** + A**[right]** = 4. Since 7 > s (=2), **right** would be **decreased**.
left = 1, right = **2**, A**[left]** + A**[right]** = 3. Since 7 > s (=2), **right** would be **decreased**.
For left = 1 & right = 1, since left == right, it would return **F**alse

CODE:

```
def solve(A):
    A.sort()
    left = 0
    right = len(A) - 1
    target = 5

    while left < right:
        _sum = A[left] + A[right]
        if _sum > target:
            right -= 1
        elif _sum < target:
            left += 1
        elif _sum == target:
            return True
    return False

if __name__ == "__main__":
    A = [2, 5, 1, 9, 6, 7]
    print(solve(A))
```

Time Complexity: O (nlogn)

Space Complexity: O(1)

3rd Approach – Hashing

A = [2, 3, 1, 4, 6]

In an empty dictionary, the first element 2 is not present in the dict so we will add 2 to the dictionary and we need to search for 7 – 2 = 5.

Now, the second element 3 is not present in the dict, so we will add 3 to the dictionary. It can be true is we already have visited a value 7 – 3 = 4.

Now the third element 1 is not present in the dict, so we will add 3 to the dictionary. It can be true is we already have visited a value 7 – 1 = 6.

At value 4, 7 – 4 = 3, since 3 is already present in the dict, the program will return **T**rue.

If the array is exhausted and it does not find the pair then it would return **F**alse.

CODE:

```
def solve(A, target):
    no_map = dict()
    for x in A:
        if target - x in no_map:
            return True
        no_map[x] = True
    return False

if __name__ == "__main__":
    A = [2, 3, 1, 4, 6]
    print(solve(A, 15))
```

https://www.geeksforgeeks.org/hashing-data-structure/