**HASHING QUESTIONS**

Table of Contents

[**Theory** 1](#_Toc186046135)

[What is hashing? 2](#_Toc186046136)

[What are components of hashing? 2](#_Toc186046137)

[Use cases of hashing and its use in system design. 3](#_Toc186046138)

[How hashMap/ dict/ defaultdict is implemented internally? 4](#_Toc186046139)

[LEVEL 1: **Easy** 5](#_Toc186046140)

[1. Write a Python function to find the first non-repeating character in a string using a hash table. 5](#_Toc186046141)

[2. Given two arrays A and B, you need to check if arrays are equal or not. Two arrays are equal if both contains same set of elements with same count, though they may be have different arrangement . 5](#_Toc186046142)

[3. Two sum problem. Given an array of integers and a target, return the indices of two numbers in array which add up to the target. All integers in the array occur only once. 5](#_Toc186046143)

[4. Given two arrays and given a value target. The problem is to count all pairs comprising of one element from each array such that they add up to target value. 5](#_Toc186046144)

[5. Write a function to check if two strings are anagrams (anagrams are words that can be formed by rearranging letters of each other). 5](#_Toc186046145)

[6. Given an array of integers, find the count of pair of integers whose sum gives 0 reminder when divided by 5. 5](#_Toc186046146)

[LEVEL 2: **Medium** 6](#_Toc186046147)

[1. Find length of longest subarray whose sum is 0. 6](#_Toc186046148)

[2. Given an. 6](#_Toc186046149)

[LEVEL 3: **Difficult** 7](#_Toc186046150)

[1. Write a Python function to find count of subarrays whose xor value is m. 7](#_Toc186046151)

[**SOLUTIONS:** 8](#_Toc186046152)

[**LEVEL 1:** 8](#_Toc186046153)

[**LEVEL 2:** 12](#_Toc186046154)

# **Theory**

## What is hashing?

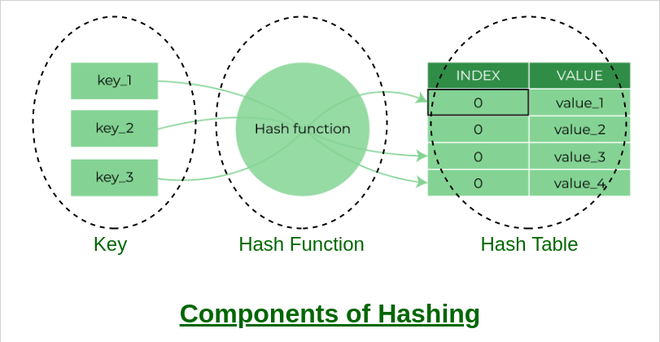
Hashing refers to the process of taking input data and generating a fixed-size string of characters, which is usually of a shorter length than the original data. This fixed-size string is known as the hash value or hash code. The purpose of hashing is to efficiently map data of arbitrary size to a fixed-size value.

* Hashing is basically mapping of large set of data to small set of data.

## What are components of hashing?

1. **Input Data:** This refers to the original data that needs to be hashed. It can be any type of data, such as a string, file, or object.
2. **Hash Function:** A hash function is a mathematical algorithm that takes the input data and produces a fixed-size string of characters known as the hash value or hash code. It determines how the input data is transformed into the hash value.
3. **Hash Value:** The hash value is the output of the hash function. It is a fixed-size string that represents the input data in a condensed form. The hash value is typically of a shorter length than the original data.
4. **Hash Table:** In some applications, a hash table or hash map is used to store the hash values and associate them with corresponding data. A hash table allows for efficient retrieval and storage of data based on the hash values.
5. **Collisions:** Collisions occur when different input data produces the same hash value. Collisions are an inherent aspect of hashing due to the possibility of mapping multiple inputs to the same output. Collision resolution techniques are employed to handle and minimize collisions.

Collisions can be resolved using various techniques such as chaining (using linked lists to store multiple values with the same hash value) or open addressing (probing for an alternative empty slot).



## Use cases of hashing and its use in system design.

1. **Data Storage and Retrieval:** Hashing allows for quick indexing and searching in data structures like hash tables. Hashing enables constant-time access to data by generating unique hash values that serve as keys for storing and retrieving information.
2. **Caching:** Hashing is utilized in caching mechanisms to improve system performance. By using hash values as cache keys, data can be quickly accessed from memory or a cache store instead of retrieving it from slower storage systems. This helps reduce latency and speeds up response times in systems with high read-intensive workloads.
3. **Load Balancing:** Hashing can be employed for distributing requests or workload across multiple servers or nodes in a load-balanced system. By generating hash values from the request data, the system can determine which server or node should handle the request. This ensures even distribution of workload and prevents overloading of specific resources.
4. **Data Deduplication:** Hashing is used in systems that require data deduplication, such as backup and storage systems. Hash values are computed for data chunks, and duplicate chunks are identified by comparing their hash values. This allows for efficient storage of unique data chunks while eliminating redundant copies, saving storage space and reducing bandwidth requirements.
5. **Password Storage:** Hashing plays a crucial role in secure password storage. Instead of storing actual passwords, their hash values are stored. When a user enters a password for authentication, the system compares the hash value of the entered password with the stored hash value. This approach enhances security by ensuring that even if the password database is compromised, the actual passwords remain concealed.
6. **Digital Signatures:** Hashing is a key component of digital signature algorithms. A hash function is used to generate a hash value from the message being signed. This hash value is then encrypted with the private key of the sender to create a digital signature. Verifiers can use the corresponding public key to decrypt the signature and compare it with the hash value of the received message, ensuring its authenticity and integrity.
7. **URL Shortener:** Hash tables allow for fast lookup and retrieval of shortened URLs based on a unique identifier or key. The original URL is hashed, generating a shorter key or hash value. This hash value serves as the key to store the original URL in the hash table.

**NOTE**

For python hashing code, use default dict instead of dictionary to make hash table. Since if we retrieve unavailable key at hash table, it will show error in dict. While in default dict it shows 0 or any value we set as default.

## How hashMap/ dict/ defaultdict is implemented internally?

Internally hashMap or dictionary is implemented using array and linked list. , a hash map (or hash table) is typically implemented using an array of buckets, where each bucket can store multiple key-value pairs. The implementation of a hash map involves the following steps:

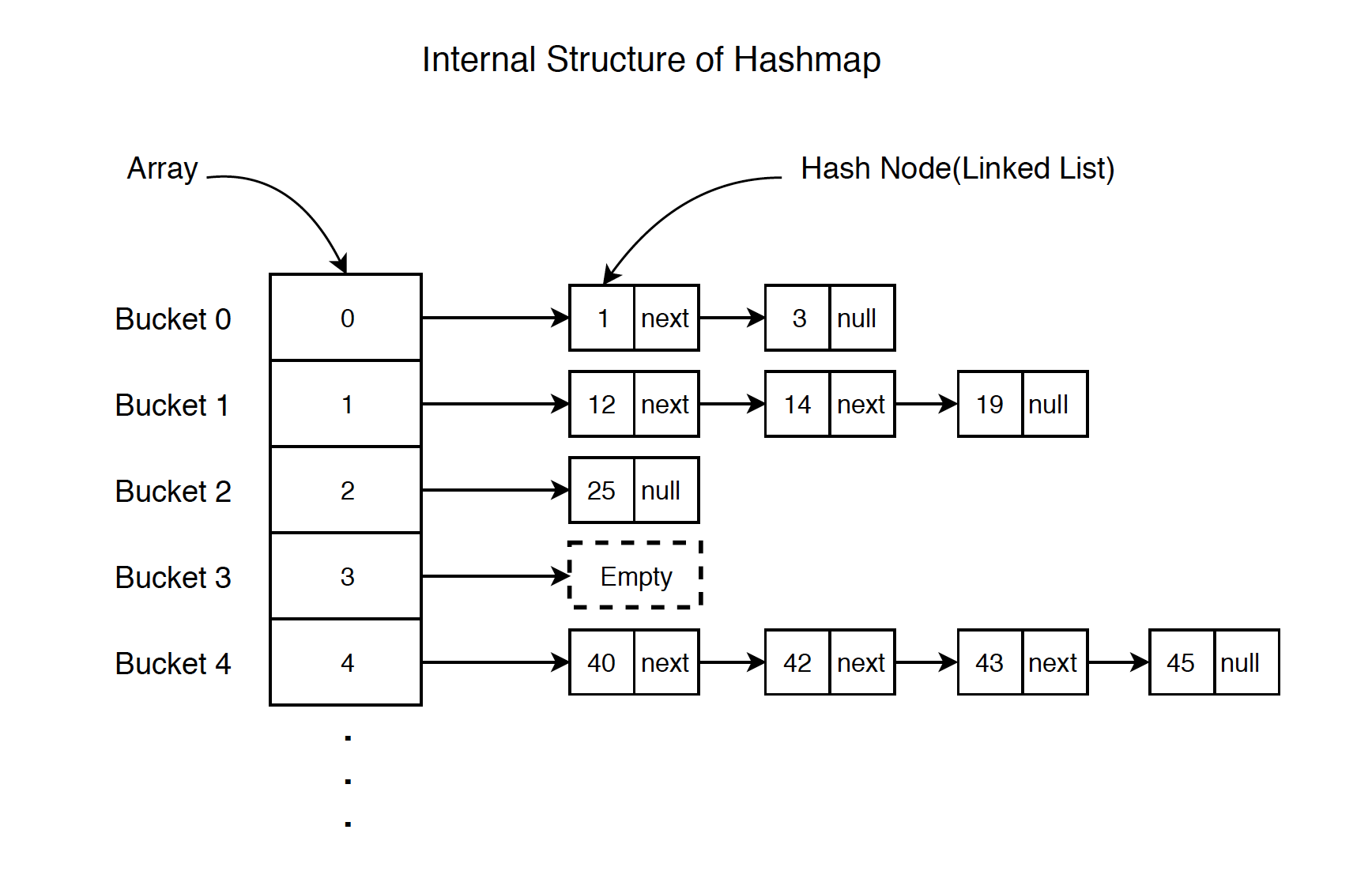
**Hash Function:** A hash function is used to convert the key into an index value that corresponds to a specific bucket in the array. The hash function should distribute the keys uniformly across the array to minimize collisions and provide efficient access to the stored values.

**Bucket Array:** The hash map maintains an array of buckets, where each bucket can store multiple key-value pairs

**Collision Handling:** Collisions occur when two or more keys are hashed to the same index in the bucket array. Various collision resolution techniques can be used, such as separate chaining or open addressing.

**Key-Value Storage:** The key-value pairs are stored in the appropriate bucket based on their hashed index value. When inserting a new key-value pair, the hash map calculates the hash value of the key, determines the corresponding bucket index, and stores the pair in that bucket.

**Retrieval and Updates:** To retrieve a value, the hash map calculates the hash value of the given key, determines the bucket index, and then searches for the key within the bucket. If the key is found, the associated value is returned. For updating or deleting a key-value pair, a similar process is followed to locate and modify the specific pair



## Python defaultdict

The functionality of both dictionaries and defaultdict is almost the same except for the fact that defaultdict never raises a KeyError. It provides a default value for the key that does not exist.

**Syntax:** defaultdict(default\_factory)

default\_factory: A function returning the default value for the dictionary defined. If this argument is absent then the dictionary raises a KeyError.

Eg 1: Get count of all elements in array

from collections import defaultdict

# Defining the dict

d = defaultdict(int)

L = [1, 2, 3, 4, 2, 4, 1, 2]

# Iterate through the list for keeping the count

for i in L:

    # The default value is 0

    d[i] += 1

print(d)

Eg 2: Using Lambda function

from collections import defaultdict

#can use lambda function to assign default value

dd = defaultdict(lambda:-1)

print(dd[2])

Eg 3: Value as list

from collections import defaultdict

d = defaultdict(list)

for i in range(5):

    d[i].append(i)

print(d)

#{0: [0], 1: [1], 2: [2], 3: [3], 4: [4]}

# LEVEL 1: **Amateur**

1. <https://leetcode.com/problems/number-of-good-pairs/?envType=problem-list-v2&envId=hash-table>

*#Approach: maintain count of how many same numbers came,*

*#Number of pairs will be count equal to count of numbers that came before it*

class Solution:

    def numIdenticalPairs(*self*, *nums*: List[int]) -> int:

*#Since constraints are : 1 <= nums[i] <= 100*

        hashArr = [0]\*101

        count=0

*for* num *in* *nums*:

            count += hashArr[num]

            hashArr[num]+=1

*return* count

### Write a Python function to find the first non-repeating character in a string using a hash table.

### Two sum problem. Given an array of integers and a target, return the indices of two numbers in array which add up to the target. All integers in the array occur only once.

### Given two arrays and given a value target. The problem is to count all pairs comprising of one element from each array such that they add up to target value.

### Write a function to check if two strings are anagrams (anagrams are words that can be formed by rearranging letters of each other).

### Given an array of integers, find the count of pair of integers whose sum gives 0 reminder when divided by 5.



<https://www.geeksforgeeks.org/problems/array-subset-of-another-array2317/1?page=1&category=Hash&difficulty=Basic&sortBy=difficulty>

*#User function Template for python3*

*from* collections *import* defaultdict

class Solution:

*#Function to check if a is a subset of b.*

    def isSubset(*self*, *a*, *b*):

*# Your code here*

        dd\_a = defaultdict(int)

        dd\_b = defaultdict(int)

*for* ele *in* *a*:

            dd\_a[ele]+=1

*for* ele *in* *b*:

            dd\_b[ele]+=1

*for* key *in* dd\_b.keys():

*if* dd\_b[key] > dd\_a[key]:

*return* False

*return* True

<https://www.geeksforgeeks.org/problems/substrings-with-similar-first-and-last-characters3644/1?page=1&category=Hash&difficulty=Basic&sortBy=difficulty>

*#User function Template for python3*

*from* collections *import* defaultdict

class Solution:

    def countSubstringWithEqualEnds(*self*, *s*):

*# code here*

        dd = defaultdict(int)

        count=0

*for* i *in* range(len(s)):

            dd[s[i]]+=1

*for* key *in* dd.keys():

            count += dd[key] \* (dd[key]+1)//2

*return* count

2.

<https://www.geeksforgeeks.org/problems/distinct-elements-in-a-stream1557/1?page=1&category=Hash&difficulty=Basic&sortBy=difficulty>

*from* collections *import* defaultdict

class Solution:

    def getDistinct(*self*, *arr*):

*#code here*

        hashArr = defaultdict(int)

        ans=[0]\*len(*arr*)

        set1 = set()

*for* i *in* range(len(*arr*)):

*if* *arr*[i]>0:

*if* *arr*[i] not in set1:

                    set1.add(*arr*[i])

                hashArr[*arr*[i]]+=1

*else*:

                temp = -1\**arr*[i]

*if* temp in set1:

                    hashArr[temp]-=1

*if*(hashArr[temp]==0):

                        set1.remove(temp)

            ans[i] = len(set1)

*return* ans

<https://www.geeksforgeeks.org/problems/anagram-palindrome4720/1?page=2&category=Hash&difficulty=Basic&sortBy=difficulty>

*#User function Template for python3*

*from* collections *import* defaultdict

class Solution:

    def isPossible(*self*, *S*):

        dd = defaultdict(int)

*for* ele *in* *S*:

            dd[ele]+=1

        count\_odd=0

*for* key *in* dd.keys():

*if* dd[key]%2==1:

                count\_odd+=1

*return* 1 *if* count\_odd<=1 *else* 0

*# code here*

# LEVEL 2: **Medium**

### Find length of longest subarray whose sum is 0.

### Given an.

https://leetcode.com/problems/longest-consecutive-sequence/description/

*#Approach: store all num in set*

*#now loop all nums, if given number don't have any number smaller than that, means it can be start of sequence,start iterating from that number untill we have number+1 in it*

class Solution:

    def longestConsecutive(*self*, *nums*: List[int]) -> int:

        num\_set = set(*nums*)

        max\_len=0

*for* num *in* num\_set:

*if* (num-1) not in num\_set:

                temp=num

                temp\_len=0

*while* temp+1 in num\_set:

                    temp\_len+=1

                    temp = temp+1

*#+1 is to include max num which don't have num+1*

                max\_len = max(max\_len,temp\_len+1)

*return* max\_len

# LEVEL 3: **Difficult**

### Write a Python function to find count of subarrays whose xor value is m.

# **SOLUTIONS:**

## **LEVEL 1:**

1. Non repeating char

def find\_first\_non\_repeating\_char(input\_string):

    char\_count = {}

    for char in input\_string:

        if char in char\_count:

            char\_count[char] += 1

        else:

            char\_count[char] = 1

    for char in input\_string:

        if char\_count[char] == 1:

            return char

    return None

*# Example usage:*

input\_string = "aabbcde"

result = find\_first\_non\_repeating\_char(input\_string)

print(result)

*#Using default Dict*

from collections import defaultdict

def non\_rep (s):

    chars = defaultdict(int)

    for i in s:

        chars[i] +=1 *#Can't do with dict as if i don't exist in dict it throw KeyError*

    for i in s:

        if chars[i]==1:

            return i

    return None

input\_string = "aabbcde"

result = non\_rep(input\_string)

print(result)

1. Two sum

from collections import defaultdict

def two\_sum(arr,target):

    ans =[]

    d = defaultdict(int) *# key:value = number:index*

    for i in range(len(arr)):

        d[arr[i]]=i

    for i in d:

        if target-i in d:

            ans.append((d[i],d[target-i]))

    return ans

ans = two\_sum([2,3,6,1,5,4],7)

print(\*ans)

1. Count Pair sum

from collections import defaultdict

class Solution:

    def countPairs(self,arr1, arr2, target):

        ans\_count=0

        d1,d2=defaultdict(int),defaultdict(int) *#key:element = value:count*

        for i in arr1:

            d1[i]+=1

        for i in arr2:

            d2[i]+=1

        for i in d1:

            if(d2[target-i]):

                ans\_count += d1[i]\*d2[target-i] *#eg in arr1 2 is 2 times and in arr2*

*#8 is 2 times so 2\*2=4 pairs*

        return ans\_count

obj=Solution()

print(obj.countPairs([1,2,2,4],[3,4,6,8,8],10))

1. Anagrams

T(C) = O(n) and Space complexity = O(1)

String can be of size = 10\*\*5, but number of unique characters will still be limited, hence space is equal to size of array = max (number of unique characters) = O(1)

from collections import defaultdict

def anagrams(s1,s2):

    if len(s1) != len(s2):

        return False

    d1, d2 = defaultdict(int), defaultdict(int) *#ket,value = letter,occurence*

    for i in range(len(s1)):

        d1[s1[i]] +=1

        d2[s2[i]] +=1

    for i in d1:

        if d1[i]!=d2[i]:

            return False

    return True

print(anagrams("algorithm","logarithm"))

1. Pair sum divisible by 5

count=0

a= [1,121,2,43,534,322423411,5144,19,5,10]

d = defaultdict(int)

*#defining a hash function*

*#input value = num*

*#hash value = num%5*

def hash(num):

    return num%5

*#if num1 gives reminder 4 and num2 gives reminder 1 then num1+num2 will give reminder 0*

*#for reminder 0 count all possible combinations*

for i in a:

    d[hash(i)]+=1

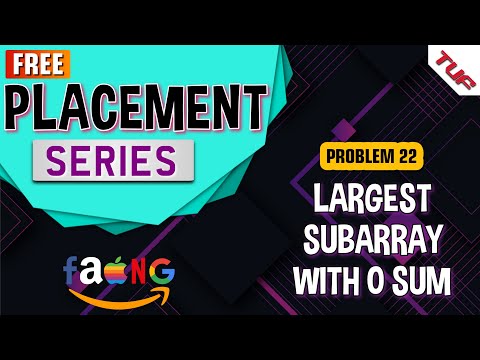
count = d[1]\*d[4] + d[2]\*d[3] + d[0]\*(d[0]-1)//2

print(count)

## **LEVEL 2:**

1. Longest subarray with zero sum

Explanation: The code maintains a cumulative sum (curr\_sum) while iterating through the array. It uses a hash set (sum\_map) to store the cumulative sum as the key and its index as the value. If the current sum is zero, it means there is a subarray starting from index 0 with a sum of zero. If the current sum is already present in the hash set, it means there is a subarray between the indices sum\_map[curr\_sum] and i (inclusive) with a sum of zero. The length of this subarray is calculated and compared with the maximum length. Finally, the maximum length is returned as the result.

[](https://www.youtube.com/watch?v=xmguZ6GbatA)

from collections import defaultdict

def longest\_subarray\_with\_zero\_sum(nums):

*# Stores the cumulative sum as the key and its index as the value*

    sum\_map = defaultdict(int)

    max\_len = 0

    curr\_sum = 0

    for i, num in enumerate(nums):

        curr\_sum += num

        if curr\_sum!=0 and curr\_sum not in sum\_map:

            sum\_map[curr\_sum] = i

        else:

            max\_len = max(max\_len,i-sum\_map[curr\_sum])

    return max\_len

a = [1,-1,3,2,-2,-8,1,7,10,23]

print(longest\_subarray\_with\_zero\_sum(a))

1. #temp

def subsets(i,n,array,s):     *#s= output so far*

    if i==n:

        print('['+s+']')

        return

    subsets(i+1,n,array,s+array[i])

    subsets(i+1,n,array,s)

array=["A","B","C"]

subsets(0,len(array),array,"")