M T W T F S S

Page No.:

Date:

YOUVA

#Hashmaps And Hashset

(2) LeetCode Q.No. (2442) { Count Number of Distinct Integers After Reverse Operations. }

Ex:-

arr = { 12, 24, 36, 41 }

arr = { 12, 24, 36, 41, 21, 42, 63, 14 }.

Ans => (8)

arr = { 13, 24, 31, 12 }.

arr = { ~~13~~, 24, ~~31~~, 12, ~~31~~, 43 ~~13~~, 21 }

Ans => (6) Distinct Count.

arr = { 2, 2, 2 } ⟶ arr = { 2, 2, 2, 2, 2, 2 } ⟶ Ans => (1)

=> Code:-

```
public int reverse(int n){
    int r = 0;
    while(n != 0){
        r = r * 10 + n%10;
        n /= 10;
    }
    return r;
}
```

T.C => O(n)

S.C => O(n).

```
public int countDistinctInteger(int[] nums){
    HashSet<Integer> set = new HashSet<>();
    for(int i = 0 ; i< nums.length ; i++){
        set.add(num[i]);
        set.add(reverse(num[i]));
    }
    return set.size();
}
```

M	T	W	T	F	S	S

Page No.: YOUVA
Date:

② LeetCode Q.No. (2744) { Find Max^m Number of String Pairs. }

Ex 1- arr = { cd, ac, dc ✓, zt, ca ✓, tz }

Count = ~~0~~ ~~1~~ 2

tz	zt
~~cd~~	~~ac~~

Ans => 2

T.C => O(n*l).

S.C => O(n*l)

where l is avg. length of String

=> Code :-

```
public int ~~s~~StringPairs(~~int[]~~ String[] words){
    HashSet<String> set = new HashSet<>();
    int count = 0;
    for(int i = 0; i < words.length; i++){
        String rev = reverse(word[i]);
        if(set.contains(rev)){
            Count++;
            set.remove(rev);
        }
        else set.add(word[i]);
    }
    ~~Sb~~ ~~el~~se return Count;
}
```

```
public String reverse(String str){
    StringBuilder sb = new SB(str);
    sb.reverse();
    return sb.toString();
}
```

③ LeetCode Q.No. (242) { Valid Anagram }

Ex:- $\underbrace{s = \text{"Raghav"} \quad t = \text{"avangR"}}_{\text{True}}$

Brute Force.
↳ Sort both Strings.
S = Raaghv t = Raaghv.
=> Equate ~~two~~ them.

Using HashMap.
map<Char, freq>;
S = "anagram"
~~S = "Ragha~~
t = "nagaram"

map1	map2
m, 1	m, 1
r, 1	r, 1
g, 1	g, 1
n, 1	a, 3
a, 3	n, 1

Now equate them. using for each loop.

=> Code :-

```
public boolean isAnagram (String s, String t){
  if(s.length() != t.length()) return false;
  HashMap<Character, Integer> map1 = new HashMap<>();
  HashMap<Character, Integer> map2 = new HashMap<>();
       //Insert in Map1
  for (int i = 0; i< s.length(); i++){
        char key1 = s.charAt(i);
        if (map1.containsKey (key1)){
              int freq = map1.get (key1);
              map1.put(key1, freq+1);
        }
        else    map1.put(key1, ~~$~~ 1);
  }
      // Insert in Map2
  for(int i = 0; i< t.length(); i++){
        Char key2 = t.CharAt(i);
        if (map2.containskey (key2)){
                  freq
              int ~~val~~ = map2.get(key2);
              map2.put(key2, freq+1);
        }
        else    map2.put(key2, 1);
  }
     // Equate them
  for (Char key : map1.keySet()){
        if(!map2.containsKey(key)) return false;
        int Val1 = map1.get(key);
        int Val2 = map2.get(key);
        if(val1 != Val2)
              return false;
  }
  return true;
}
```

(4) LeetCode Q.No. (1) { Two Sum. }

Ex :- arr = { 2 , (5) , 9 , (4) }. (indices 0 1 2 3.) target = 9.

# Brute Force → T.C ⇒ $O(n^2)$
└→ S.C ⇒ $O(1)$

Ans ⇒ { 1, 3 }

# HashMap → T.C = $O(n)$.
└→ S.C = $O(n)$.

⇒ Map < key , Val >
ele idx

arr = { 2, 5, 9, 4, 1 }. (indices 0 1 2 3 4) target = 10.

⇒ Code :-

```
public int[] twoSum(int[] nums, int target){
|    int[] ans = {-1, -1};
|    HashMap<Integer, Integer> map = new HashMap<>();
|    for(int i=0 ; i< nums.length ; i++){
|    |    int remaining = target - nums[i];
|    |    if(map.containsKey(remaining)){
|    |    |    int val = map.get(remaining);
|    |    |    ans[0] = val;
|    |    |    ans[1] = i;
|    |    |    break;
|    |    }
|    |    else map.put(nums[i], i);
|    }
|    return ans;
}
```

(4, 3)
(9 , 2)
(5 , 1)
(2 , 0)

Map .

remaining = target - arr[i];
↓
[Check is it in hashmap.]

(5) LeetCode Q.No. (1207) { Unique Number of Occurrences. }

Ex :- (1) arr = { 1, 2, 2, 1, 1, 3 }
└Ans→ True

⇒ Occurrence of 1 is (3).
⇒ Occurrence of 2 is (2).
⇒ Occurrence of 3 is (1)
} Occurrences ~~They~~ are Unique

Ex (2) :- arr = { 1, 2 }
└→ Ans ⇒ False

⇒ Occurrence of (1) is (1)
⇒ Occurrence of (2) is (1)
} Occurrences are Not Unique

# Hint ⇒ Use Both HashMap & HashSet

=> Code :- public boolean uniqueOccurrences(int[] arr){

```
HashMap<Integer, Integer> map = new HashMap<>();
for(int i=0 ; i< arr.length ; i++){
        int key = arr[i];
        if(map.containsKey(key)){
            int freq = map.get(key);
            map.put(key, freq+1);
        }
        else  map.put(key, 1);
}.
HashSet<Integer> set = new HashSet<>();
for(int val : map.values()){
        if(set.contains(val)) return false;
        else set.add(val);
}
return true;
}.
```

⑥ LeetCode Q.No. (2094) { Finding 3-Digit Even Numbers }

Ex① :- digits = {2, 1, 3, 0} $\xrightarrow{\text{Ans :-}}$ Ans = {102, 120, 130, 132, 210, 230, 302, 310, 312, 320.}

Ex② :- digits = {2, 2, 8, 8, 2}

↳ Ans → Ans = {222, 228, 282, 288, 822, 828, 882}

1st :- Create HashMap ~~of~~ of arr value & their frequency

Code :- public int[] findEvenNumbers(int[] digits);

```
HashMap<Integer, Integer> map = new HashMap<>();
for(int i=0 ; i< digits.length ; i++){
    int key = digits[i];
    if(map.containskey(key)){
        int freq = map.get(key);
        map.put(key, freq+1);
    }
    else map.put(key, 1);
}
```

T.C = O(n + 500)

```
List< Integer> ans = new ArrayList<>();
for(int i = 100; i < 1000; i+=2){
    int x = i;
    int c = x%10;    x/=10;
    int b = x%10;    x/=10;
    int a = x;
    if(map.containsKey(a)){
        int aFreq = map.get(a);
        map.put(a, aFreq-1);
        if(aFreq == 1) map.remove(a);
        if(map.containsKey(b)){
            int bFreq = map.get(b);
            map.put(b, bFreq-1);
            if(bFreq == 1) map.remove(b);
            if(map.containsKey(c)){
                ans.add(i);
            }
            map.put(b, bFreq);
        }
        map.put(a, aFreq);
    }
}
```

⇒ Convert Array List to Array & Return.

⑦ LeetCode Q.No.(3) { Longest Substring without Repeating Characters. }

Ex(1):-

indices: 0 1 2 3 4 5 6 7 8 9

s = a b c a b c g b a

i = 0, j = 0.

maxlen = ~~0~~ ~~3~~ 4

len = j - i

```
=> while(j<n){
    Char ch = s.charAt(j);
    if(map.containsKey(ch) && map.get(ch) >= i){
        int len = j - i;
        maxlen = Math.max(maxlen, len);
        while(s.charAt(i) != ch) i++;
        i++;
    }
    map.put(ch, j);
    j++;
}
len = j - i;   maxlen = Math.max(maxlen, len);
```

(a, 8)
(b, 7)
(g, 6)
~~(c, 2)~~ (c, 5)
~~(b, 1)~~ ~~(b, 4)~~
~~(a, 0)~~ ~~(a, 3)~~

(8) LeetCode Q.No. (1497) { Check if Array pairs are divisible by k. }

Ex :- arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 } k = 5

→ (1, 9), (2, 8), (3, 7), (4, 6), (5, 10)

→ (1, 4), (2, 3), (6, 9), (7, 8), (5, 10)

2 + 3 => 5

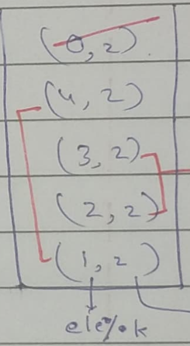
a & b & I need to figure out if $(a + b) \% k == 0$.

→ $a \% k + b \% k == 0$

$(-a) \% b = -[a \% b]$

arr = { 1, 2, 3, 4, ~~5~~, 6, 7, 8, 9, 5, 10 } k = 5

arr[i] % k → 1, 2, 3, 4, 1, 2, 3, 4, 0, 0



Steps :-

→ first Create a map & Store every ele % k & there frequency in map.

→ if ele % k ~~≤~~ 0 (-ve) ~~then~~ then add extra k in it & then add it to map.

Special Cases:

→ if map contain '0' then check it's freq. if it is odd return false. if even then remove that pair from map. (0, 2)

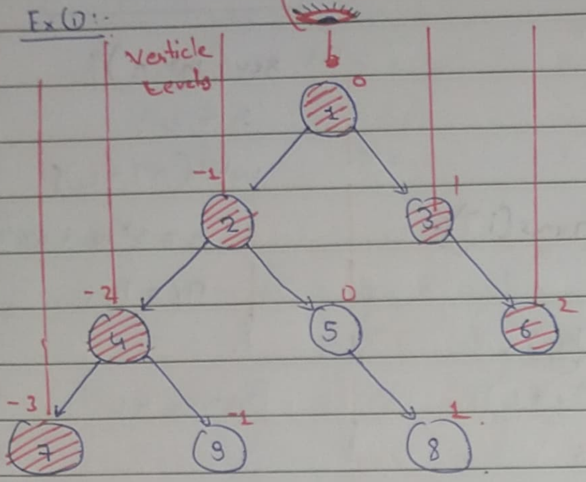
→ ~~if map Contain (k/2)~~ if k is even & map contains (k/2) ele then check its freq if it's add return false. else remove it from map.

→ Now iterate on map & Calculate remaining (rem) value of every key. if map not contain rem then return false. if it Contains it then check freq of of key & rem if freq are not equal return false.

→ After completition of iteration return true.

(9) Hacker Rank { Top View of Binary Tree. }

Ex(1):-



# Use Level Order traversal Iterative method.

Steps:- → Create map & queue.
map will store vertical level & value
Queue will store pair of level ~~node~~ & nodes.

→ Run loop ~~[illegible]~~

while (q.size() > 0) {
→ 1st remove pair & store lvl & Node in new variable.
→ store minLevel & maxlevel also ~~to~~
→ if ~~Root~~ map not contain lvl then add it to map (lvl & Node value ~~node~~).
→ if node.left != null add lvl-1 & node.left to queue.
→ if node.Right != null add lvl+1 & node.right to queue.
}.

(-3, 7)
(2, 6)
(-2, 4)
(1, 3)
(-1, 2)
(0, 1)

Map

This is Ans

for Bottom View. Dont Check this Condition. every time you need to Add lvl & value to map.

~~(1, 0)~~ ~~(2, -1)~~ ~~(3, 1)~~ ~~(4, -2)~~ ~~(5, 0)~~
~~(6, 2)~~ ~~(7, -3)~~ ~~(9, -1)~~ ~~(8, 1)~~

Queue

→ Run for (minlevel to maxlevel) {
sout (map.get(i) + " ");
}

(1, 0) (2, -1) (3, 1) (4, -2)

Node ← (1, 0) → lvl

(5, 0) (6, 2) (7, -3) (9, -1) (8, 1)

(5, 0) → Already Present. (9, -1), (8, 1) → Already Present

(10) Leetcode Q. No. (1814) { Count Nice Pairs in An Array }

Nice Pairs ⇒ nums[i] + rev(nums[j]) == nums[j] + rev(nums[i]);
→ nums[i] − rev(nums[i]) == nums[j] − rev(nums[j]);

Ex(1) nums = {42, 11, 1, 97} —Ans→ (2)

→ 42 + rev(97) = 97 + rev(42) ⇒ 42+79 == 97+24.
(121) (121)

→ 11 + rev(1) = 1 + rev(11) ⇒ 11 + 1 == 1 + 11
(12) (12)

=> Code :-

arr = { 13, 10, 35, 24, 76 }
-18, 9, -18, -18, 9.
$\boxed{13 - 31} \Rightarrow -18$

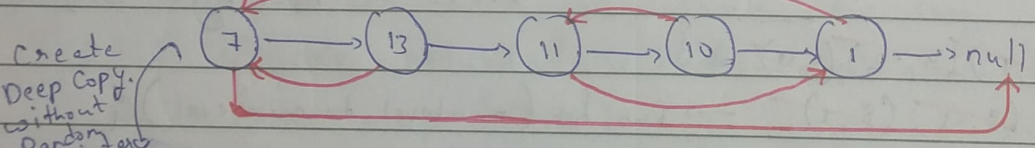
| (9, 2) |
| (-18, 3) |
Map.

Count :- ~~0~~ ~~1~~ ~~3~~ 4

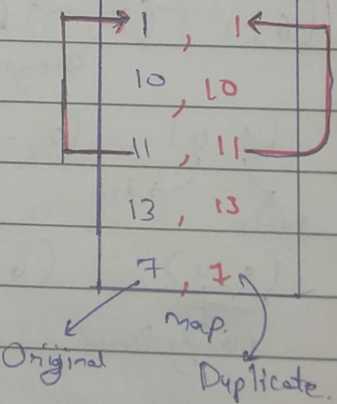
```
int CountNicePairs(nums) {
    Count = 0;
    n = nums.length;
    ~~new~~ map<Integer, Integer> ;
    for( i = 0   to   i < n  ){
        key = nums[i] - rev(nums[i]);
        if(map.contain(key)){
            ~~freq~~ freq = map.get(key);
            map.put(key, freq + 1);
            Count += freq;
            Count % = 1000000007;
        }
        else map.put(key, 1);
    }
    return Count;
}
```

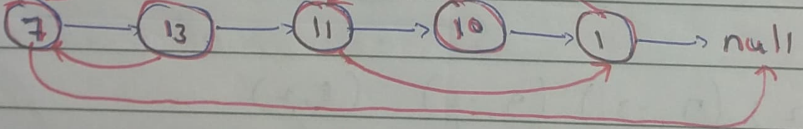
```
int rev(int n){
    r = 0;
    while(n != 0){
        r = r*10 + n%10;
        n /= 10;
    }
    return r;
}
```

(4) LeetCode Q.No. (138) { Copy List with Random Pointer ? }



T.C => O(n)
Extra Space => O(n) (map)





dup.random = map.get(ori.random);

```
for(Node key : map.keySet()){
    Node dTemp = map.get(key);
    dTemp.random = map.get(key.random);
}
```

=> 1st Create deep Copy.
=> then add Node's of both list to map.

(12) Leetcode Q No. (1830) { Unique Length - 3 Palindromic Subsequences. }

Ex ①:-

Str = " abcaaabcccb ";

=> aaa	bab	ccc
aba	bcb	cac
aca	bbb.	cbc

Ex ②:- Str = aabca

aaa
aba
aca.

Ex(3):-

```
        0 1 2 3 4 5 6 ..
Str = " a b g c a b g "
        ↓         ↓
       1st       Lst
```

{g c a} → count unique ch. using set.

fMap

(c, 3)
(g, 2)
(b, 1)
(a, 0)

LMap

(c, 3)
~~(g, 2)~~ (g, 6)
~~(b, 1)~~ (b, 5)
~~(a, 0)~~ (a, 4)

→ Store first & Last Occurence index of ever Ch

```
int Count = 0;
for(char ch : fMap.keySet()){
    int fIdx = fMap.get(ch);
    int LIdx = LMap.get(Ch);
    HashSet<Character>;
    for (i = fIdx + 1 to j < LIdx){
        set.add (s.charAt(i));
    }.
    Count += set.size();
}
return Count.
```