

Laboratory work report

Submitted by:

Madiyar Zhumabekov

IS-1812K

Submitted to:

Yeskendir Sultanov

**Laboratory work #3. Sets and Maps**

Deadline: 22nd September 2020

Solve these problems using Sets or Maps. MMM

<https://leetcode.com/problems/contains-duplicate/>

**Task:** Given an array of integers, find if the array contains any duplicates.

Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

**Answer:** I use set to find duplicates. If set contains value which exist in set, code return true. Time complexity O(N)

**My code:**

class Solution {

public boolean containsDuplicate(int[] nums) {

Set<Integer> set = new HashSet<Integer>();

for (int i : nums){

if(set.contains(i)){

return true;

}

else{

set.add(i);

}

}

return false;

}

}

<https://leetcode.com/problems/two-sum/>

**Task:** Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

**Answer:** I use set and 2 loops to find unique values, that match the requirement. An d use iterator to record values from set to array. Time complexity: O(N)

**My code:**

class Solution {

public int[] twoSum(int[] nums, int target) {

HashSet<Integer> ans = new HashSet<Integer>();

for(int i = 0; i<nums.length; i++){

for(int j = i + 1; j<nums.length; j++){

if(nums[i]+nums[j]==target){

ans.add(i);

ans.add(j);

}

}

}

int[] result = new int[ans.size()];

Iterator iterator = ans.iterator();

int i = 0;

while (iterator.hasNext()) {

result[i]=(int) iterator.next();

i++;

}

return result;

}

}

<https://leetcode.com/problems/intersection-of-two-arrays/>

**Task:** Given two arrays, write a function to compute their intersection.

**Answer:** Declared a variable HashSet. In the loop, I added variables to the HashSet that occur in both arrays. To add result to the array, I used Iterator. And in the loop, I wrote the HashSet values to the array. Time complexity: O(NM).

**My code:**

class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

HashSet<Integer> ans = new HashSet<Integer>();

for(int i = 0; i<nums1.length; i++){

for(int j = 0; j<nums2.length; j++){

if(nums1[i]==nums2[j]){

ans.add(nums1[i]);

}

}

}

int[] result = new int[ans.size()];

Iterator iterator = ans.iterator();

int i = 0;

while (iterator.hasNext()) {

result[i]=(int) iterator.next();

i++;

}

return result;

}

}

<https://leetcode.com/problems/group-anagrams/>

**Task:** Given an array of strings strs, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Answer:** I use map to sort values. Firstly I convert every string in string array to char array and sort. Then in string key I record value of characters array. If map does not have a key. Add to map key and declare new list. If in map exist key I just record string. Time complexity O(N).

**My code:**

class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

Map<String, List> map = new HashMap<String, List>();

for(String x: strs){

char[]chars = x.toCharArray();

Arrays.sort(chars);

String key = new String(chars);

if(!map.containsKey(key)){

map.put(key, new ArrayList());

}

map.get(key).add(x);

}

return new ArrayList(map.values());

}

}

<https://leetcode.com/problems/word-pattern/>

**Task:** Given a pattern and a string s, find if s follows the same pattern.

Here **follow** means a full match, such that there is a bijection between a letter in pattern and a **non-empty** word in s.

**Answer:** I use 2 maps and String array. In first if I check length of words and length of pattern. In loop I check if first map contain char ch. If not I check hm2 contain words at index I. If yes I return false, else I just add values with keys to 2 maps. If first map contain key char ch, I check whether the value is equal to the value in this key. If not I return false.

**My code:**

class Solution {

public boolean wordPattern(String pattern, String str) {

String words[]=str.split(" ");

if (words.length != pattern.length()){

return false;

}

HashMap<Character,String> hm1=new HashMap<>();

HashMap<String,Character> hm2=new HashMap<>();

for(int i=0;i<pattern.length();i++){

char ch=pattern.charAt(i);

if(!hm1.containsKey(ch)){

if(hm2.containsKey(words[i]))

return false;

else{

hm1.put(ch,words[i]);

hm2.put(words[i],ch);

}

}

else{

String val=hm1.get(ch);

if(!val.equals(words[i])){

return false;

}

}

}

return true;

}

}

<https://leetcode.com/problems/3sum/>

**Task:** Given an array nums of n integers, are there elements a, b, c in nums such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

Notice that the solution set must not contain duplicate triplets.

**Answer:** Sort the array and use set. In loop I use pointers. In while loop calculate sum of 3 elements. If the sum is equal to 0, add to the set this three elements. If sum is greater than 0 I decrease end. If sum is smaller than 0 I increase start. Then return set as Array list.

**My code:**

class Solution {

public List<List<Integer>> threeSum(int[] nums) {

Arrays.sort(nums);

Set<List<Integer>> set = new HashSet<>();

for (int i = 0; i < nums.length - 2; i++) {

int start = i + 1;

int end = nums.length - 1;

while (start < end) {

int sum = nums[i] + nums[start] + nums[end];

if (sum == 0) set.add(Arrays.asList(nums[i], nums[start++], nums[end--]));

else if (sum > 0) end--;

else if (sum < 0) start++;

}

}

return new ArrayList<>(set);

}

}

<https://leetcode.com/problems/contains-duplicate-ii/>

**Task:** Given an array of integers and an integer *k*, find out whether there are two distinct indices *i* and *j* in the array such that **nums[i] = nums[j]** and the **absolute** difference between *i* and *j* is at most *k*.

**Answer:** I use map. In loop I give the value nums[i] to the integer x. If map does not have key at x. Add to map key x. If map has key x, check difference between I and j. If difference between is smaller than k return true. Time complexity O(N).

**My code:**

class Solution {

public boolean containsNearbyDuplicate(int[] nums, int k) {

Map<Integer,Integer> last\_pos = new HashMap<>();

for(int i=0; i<nums.length; i++){

int x = nums[i];

if(last\_pos.containsKey(x)){

int l\_pos = last\_pos.get(x);

if((i - l\_pos) <= k){

return true;

}

}

last\_pos.put(x, i);

}

return false;

}

}

<https://leetcode.com/problems/random-pick-index/>

**Task:** Given an array of integers with possible duplicates, randomly output the index of a given target number. You can assume that the given target number must exist in the array.

**Note:**  
The array size can be very large. Solution that uses too much extra space will not pass the judge.

**Answer:** I use list of integers and add all index of elements which equal to target. randomly selected an index within the list.

**My code:**

class Solution {

int[] nums;

public Solution(int[] nums) {

this.nums=nums;

}

public int pick(int target) {

List<Integer> list= new ArrayList<>();

for(int i=0;i<nums.length;i++){

if(nums[i]==target){

list.add(i);

}

}

Random random= new Random();

int randomIndex=random.nextInt(list.size());

return list.get(randomIndex);

}

}

/\*\*

\* Your Solution object will be instantiated and called as such:

\* Solution obj = new Solution(nums);

\* int param\_1 = obj.pick(target);

\*/

<https://leetcode.com/problems/subarray-sum-equals-k/>

**Task:** Given an array of integers and an integer **k**, you need to find the total number of continuous subarrays whose sum equals to **k**.

**Answer:** I use map. In loop I calculate sum. If sum is equal to k, I increase count. If map does not contain key at sum-k, add new key with value +1.

**My code:**

public class Solution {

public int subarraySum(int[] nums, int k) {

int count = 0;

int sum = 0;

Map<Integer,Integer> map = new HashMap<>();

for (int i = 0; i<nums.length ; i++) {

sum += nums[i];

if ( sum == k) {

count++;

}

if ( map.containsKey( sum - k)) {

count += map.get(sum - k);

}

map.put(sum, map.getOrDefault(sum,0) +1);

}

return count;

}

}

<https://leetcode.com/problems/4sum-ii/>

**Task:** Given four lists A, B, C, D of integer values, compute how many tuples (i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.

To make problem a bit easier, all A, B, C, D have same length of N where 0 ≤ N ≤ 500. All integers are in the range of -228 to 228 - 1 and the result is guaranteed to be at most 231 - 1.

**Answer:**

**My code:**

<https://leetcode.com/problems/happy-number/>

**Task:** Write an algorithm to determine if a number n is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it **loops endlessly in a cycle** which does not include 1. Those numbers for which this process **ends in 1** are happy numbers.

Return True if n is a happy number, and False if not.

**Answer:** I use set and add integer n. Use while loop until n will be equal to 1. In second while loop we calculate sum. Then if set contains n which equal res return false. It means the number is not happy. Else just add n to the set.

**My code:**

class Solution {

public boolean isHappy(int n) {

HashSet<Integer> set = new HashSet<>();

set.add(n);

while(n!=1){

int res = 0;

while(n>0){

res+=(n%10)\*(n%10);

n/=10;

}

n=res;

if(set.contains(n)){

return false;

}

else{

set.add(n);

}

}

return true;

}

}

<https://leetcode.com/problems/top-k-frequent-elements/>

**Task:** Given a non-empty array of integers, return the ***k*** most frequent elements.

**Answer:**

**My code:**