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

Deadline: 22nd September 2020

Solve these problems using Sets or Maps.

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

class Solution {

public boolean containsDuplicate(int[] nums) {

Set<Integer> mySet = new HashSet<>(nums.length){

for(int x: nums){

if(mySet.contains(x)) return true;

mySet.add(x);

}

return false;

}

}

There is time complexity is O(n).

Firstly, create the Hah table set the name mySet. It contains only integer and length equal to the array nums length. The loop populates for x in nums element. The loop run with x and the condition the set contains x the program will show true else it add to the set, then loop repeat. If the set not duplicated the program show false.

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

class Solution {

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

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

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

int second = target - nums[i];

if(result.containsKey(second)){

return new int[] {result.get(second), i};

}

result.put(nums[i], i);

}

throw new IllegalArgumentException("No two sum solution");

}

}

The program iterates and insert elements into the table, also look back to check if current element’s second already exists in the table. If it exists, the program found a solution and return it keys.

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

class Solution {

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

HashSet<Integer> set1 = new HashSet<Integer>();

HashSet<Integer> set2 = new HashSet<Integer>();

for(int x: nums1) set1.add(x);

for(int n: nums2) set2.add(n);

set1.retainAll(set2);

int [] result = new int[set1.size()];

int count = 0;

for(int s: set1) result[count++] = s;

return result;

}

}

Convert both arrays to the sets. There is built-in intersection of two sets. Then convert the intersection of two sets to the array.

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

class Solution {

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

if(strs.length == 0) return new ArrayList();

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

for(String i : strs){

char[] word = i.toCharArray();

Arrays.sort(word);

String key = String.valueOf(word);

if(!result.containsKey(key))

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

result.get(key).add(i);

}

return new ArrayList(result.values());

}

}

Time complexity is O(nklogk)

If the length of the string is 0, the result will new empty array list. Create a map result : (String -> List) where each key K is sorted string. The loop iterates by each word, sorted and it’s equal to the key. In condition, the result map is not containing key, the key added to the map and new array list. Each value is the list of string from the initial input that when sorted, are equal to key.

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



class Solution {

public boolean wordPattern(String pattern, String s) {

HashMap map\_index = new HashMap();

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

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

return false;

for (Integer i = 0; i < words.length; i++) {

char key = pattern.charAt(i);

String word = words[i];

if (!map\_index.containsKey(key))

map\_index.put(key, i);

if (!map\_index.containsKey(word))

map\_index.put(word, i);

if (map\_index.get(key) != map\_index.get(word))

return false;

}

return true;

}

}

I have a single hash map which keeps track of the first occurrence of each character in pattern and each word in s. The loop goes through each character word pair, it inserts unseen characters from pattern and unseen words from s. The program goal is to make sure that the indices of each character and word match up. As soon as, the program finds a mismatch, program can return false.

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



class Solution {

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

if (nums.length < 3) return new ArrayList<>(); // if nums less than 3 element

Arrays.sort(nums); // sort array

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

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

int j = i + 1;

int k = nums.length - 1;

while (j < k) {

int sum = nums[i] + nums[j] + nums[k];

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

else if (sum > 0) k--;

else if (sum < 0) j++;

}

}

return new ArrayList<>(set);

}

}

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



class Solution {

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

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

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

Integer index = map.get(nums[i]);

if(index != null && i - index <= k)

return true;

map.put(nums[i],i);

}

return false;

}

}

The program iterates with loop and take index from map in the condition if index is not equal to the null and I – index at most k the result will true. Else it adds to the map element and index I. Another it will show false.

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

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);

}

}

Figure out all the indices where target number appear and add those indices to a list. Randomly pick an index of the list and return item from that index.

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

class Solution {

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

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

map.put(0, 1);

int count = 0, sum = 0;

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

sum += nums[i];

if(map.containsKey(sum - k))

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

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

}

return count;

}

}

I iterate the loop with nums array and add to the sum of each element of the array. Every time encounter new sum and new entry in the map corresponding to that sum. If the same sum occurs again, increment the count corresponding to that sum in the hashmap. For every sum determine the number of times the sum – k has occurred, the number of times a subarray with sum k has occurred upto the current index. Then increment the count by the same amount. After the complete array has been traversed, the count gives a result.

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

class Solution {

public int fourSumCount(int[] A, int[] B, int[] C, int[] D) {

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

for (int a: A)

for (int b: B)

map.put(a + b, map.getOrDefault(a + b, 0) + 1);

int count = 0;

for (int c: C)

for (int d: D)

count += map.getOrDefault(-(c + d), 0);

return count;

}

}

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

class Solution {

public boolean isHappy(int n) {

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

set.add(n);

while(n != 1){

int sum = 0;

while(n > 0){

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

n /= 10;

}

n = sum;

if(set.add(n) == false)

return false;

}

return true;

}

}

Create the hash set and add element n to the set. The iterate with while loop before n not equal to 1. The next while loop used to find the sum of the digit square. Then the hash set not added digit sum, the result false, else it will show true.

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



class Solution {

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

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

for (int i : nums){

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

}

ArrayList<int[]> arr = new ArrayList<>();

for (Map.Entry<Integer, Integer> e: map.entrySet()){

arr.add(new int[]{e.getKey(), e.getValue()});

}

Collections.sort(arr, (a, b) -> b[1] - a[1]);

int[] a = new int[k];

for(int i=0; i<k;i++){

a[i] = (arr.get(i))[0];

}

return a;

}

}

Count each element to the map. Copy all elements from hashmap to arraylist to sort it. Sort the array by descending order, based on count. Make a new array and return it in the result.