**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 set = new HashSet<>(nums.length);

for(int i : nums){

set.add(i);

}

return !(set.size() == nums.length);

}

}

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

class Solution {

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

int[] ans=new int[2];

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

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

{

int value=target-nums[i];

if(mp.containsKey(value))

{

ans[0]=mp.get(value);

ans[1]=i;

break;

}

else

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

}

return ans;

}

}

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

class Solution {

public:

vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {

set<int> s(nums1.begin(), nums1.end()), res;

for (auto a : nums2) {

if (s.count(a)) res.insert(a);

}

return vector<int>(res.begin(), res.end());

}

};

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

class Solution {

public:

vector<vector<string>> groupAnagrams(vector<string>& strs) {

vector<vector<string>> ans;

unordered\_map<string, vector<string>> m;

string sword;

for(auto word : strs)

{

sword = word;

sort(sword.begin(), sword.end());

m[sword].emplace\_back(word);

}

for(auto i : m){

ans.push\_back(i.second);

}

return ans;

}

};

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

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)){ //ch not mapped with any string.

if(hm2.containsKey(words[i])) // if that string with any other char.

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/>

class Solution {

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

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

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

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

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

int want = 0 - nums[i] - nums[j];

if (map.containsKey(want)){

if (map.get(want) != i && map.get(want) != j){

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

temp.add(nums[i]);

temp.add(nums[j]);

temp.add(want);

Collections.sort(temp);

set.add(temp);

}

}

}

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

}

return new ArrayList<>(set);

}

}

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

class Solution {

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

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

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

Integer pre = m.get(nums[i]);

if(pre != null && i - pre <= k) {

return true;

}

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

}

return false;

}

}

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

class Solution {

Map<Integer, List<Integer>> map;

Random rand = new Random();

public Solution(int[] nums) {

map = new HashMap<>();

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

map.computeIfAbsent(nums[i], k -> new ArrayList<>()).add(i);

}

public int pick(int target) {

List<Integer> positions = map.get(target);

return positions.get(rand.nextInt(positions.size()));

}

}

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

class Solution {

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

int count = 0;

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

for (int end = start; end < nums.length; end++) {

int curSum = 0;

for (int i = start; i <= end; i++) {

curSum += nums[i];

}

if (curSum == k) {

count++;

}

}

}

return count;

}}

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

class Solution {

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

int ans=0;

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

//store the number of ways to make a particular sum.

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

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

if(! hashmap.containsKey(A[i]+B[j])){

hashmap.put(A[i]+B[j],1);

}else{

int a=hashmap.get(A[i]+B[j]);

hashmap.put(A[i]+B[j],a+1);

}

}

}

//check if the target-(3rd+4th)number is present or not in hashmap

//if present add the no of ways to the ans list.

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

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

if(hashmap.containsKey(-(C[i]+D[j])))

ans +=hashmap.get(-(C[i]+D[j]));

}

}

return ans;

}

}

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

class Solution {

int max\_itter = 20; //change this to ur liking. (lower = faster)

boolean bool = false;

public boolean isHappy(int n) {

return recur(digits(n));

}

public boolean recur(int[] nums){

int num = 0;

//add up each number

for(int \_num: nums){

num+= (int)(Math.pow(\_num,2));

}

//if 1 then true

if(num == 1){

bool = true;

return bool;

}

//if max itterations didnt reach

//0 then keep on with recursion.

//this max is modifiable via line #4

max\_itter--;

if(max\_itter!=0)

recur(digits(num));

return bool;

}

public int[] digits(int num){

//total num of digits at first

int num\_digits = (int)Math.log10(num)+1;

int[] arr = new int[num\_digits];

for(int i=num\_digits-1; i>=0; i--){

//tmp divisor

int tmp = num/((int)Math.pow(10,i));

//add in reverse, as well as in the array

arr[(num\_digits-1) -i] = tmp%(10);

}

return arr;

}

}

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

class Solution {

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

if (k == nums.length) { return nums; }

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

int count = 0;

for (Integer num: nums) {

if (frequency.containsKey(num)) {

count = frequency.get(num);

frequency.put(num, count+1);

} else {

frequency.put(num, 1);

}

}

// Copy HashMap to List

List<Map.Entry<Integer, Integer>> sortedMap = new ArrayList<>(frequency.entrySet());

// Sort the list based on values of <K, V> pair

Collections.sort(sortedMap, new Comparator<Map.Entry<Integer, Integer>>() {

public int compare(Map.Entry<Integer, Integer> e1,

Map.Entry<Integer, Integer> e2) {

return (e2.getValue().compareTo(e1.getValue()));

}

});

// Find the top K frequent elements :)

int[] result = new int[k];

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

result[i] = sortedMap.get(i).getKey();

}

return result;

}

}