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

Deadline: 22nd September 2020

Solve these problems using Sets or Maps.

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

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.

**Example 1:**

**Input:** [1,2,3,1]

**Output:** true

**Code:**

bool containsDuplicate(vector<int>& nums) {

if(nums.size() < 2)

return false;

unordered\_set<int> a;

for(int num : nums){

a.insert(num);

}

if(a.size() == nums.size())

return false;

else return true;

}

**Report:**

**Set solution.**

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

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.

**Example 1:**

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

**Output:** Because nums[0] + nums[1] == 9, we return [0, 1].

**Code:**

vector<int> twoSum(vector<int>& nums, int target) {

map<int, int> mInd;

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

if (mInd.count(target - nums[i]) > 0)

return { mInd[target - nums[i]], i};

mInd[nums[i]] = i;

}

return { -1, -1 };

}

**Report:**

**Map solution.**

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

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

**Example 1:**

**Input:** nums1 = [1,2,2,1], nums2 = [2,2]

**Output:** [2]

**Code:**

class Solution

{

public:

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

{

int i;

vector <int> v1;

map <int,int> m1;

for(i=0;i<nums1.size();i++)

m1[nums1[i]]=1;

for(i=0;i<nums2.size();i++)

{

if(m1[nums2[i]]==1)

{

v1.push\_back(nums2[i]);

m1[nums2[i]]=0;

}

}

return v1;

}

};

**Report:**

**Map solution.**

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

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.

**Example 1:**

**Input:** strs = ["eat","tea","tan","ate","nat","bat"]

**Output:** [["bat"],["nat","tan"],["ate","eat","tea"]]

**Code:**

class Solution {

public:

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

vector<vector<string>> res;

map<vector<int>,vector<string>> m;

for(string word : strs){

vector<int> alph(26,0);

for(char c : word){

alph[c - 'a']++;

}

m[alph].push\_back(word);

}

for(auto const& [key,value] : m){

res.push\_back(value);

}

return res;

}

};

**Report:**

**Map solution.**

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

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.

**Example 1:**

**Input:** pattern = "abba", s = "dog cat cat dog"

**Output:** true

**Code:**

class Solution {

public:

bool wordPattern(string pattern, string str) {

map<string,int> M;

stringstream s(str);

string word;

int i=0;

int count=0;

while(s>>word)

{

count++;

string x="";

x=x+pattern[i]+'\_'; // to distinguish pattern characters and string words of length 1

if(M.find(x)==M.end())

{

M[x]=i;

}

if(M.find(word)==M.end())

{

M[word]=i;

}

if(M[x]!=M[word])

return false;

i++;

}

if(count != pattern.size())

return false;

return true;

}

};

**Report: Single map solution**

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

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.

**Example 1:**

**Input:** nums = [-1,0,1,2,-1,-4]

**Output:** [[-1,-1,2],[-1,0,1]]

**Code:**

vector<vector<int>> threeSum(vector<int>& nums) {

vector<vector<int>> res;

int N = (int)nums.size();

if (N < 3) return res;

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

vector<int> temp;

int i = -1;//i is the first element of the duplicate to avoid missing results.

while (i < N) {

i++;

if (nums[i] > 0) break;//starting from element greater than 0 doesn't make sense.

int low = i + 1;

int high = N - 1;

//two sum search

while (low < high) {

while (low < high && nums[high] + nums[low] > -nums[i]) high--;

while (low < high && nums[high] + nums[low] < -nums[i]) low++;

if (low < high && nums[high] + nums[low] == -nums[i]) {

temp.clear();

temp.push\_back(nums[i]);

temp.push\_back(nums[low]);

temp.push\_back(nums[high]);

res.push\_back(temp);

//deal with duplicates

while (nums[high-1] == nums[high]) high--;

while (nums[low+1] == nums[low]) low++;

high--;

low++;

}

}

//deal with duplicates of nums[i] as well.

while (nums[i+1]==nums[i]) i++;

}

return res;

}

**Report:**

This problem can be transformed to be the twoSum problem. So we can traverse each element in nums array, and find two elements which have the given sum. So we can use twoSum squeezing algorithm to solve it. However, we have to deal with duplicates, so after we find the twoSum, we have to avoid pushing same results by moving low and high. And we have to deal with duplicate num[i]

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

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*.

**Example 1:**

**Input:** nums = [1,2,3,1], k = 3

**Output:** true

**Code:**

bool containsNearbyDuplicate(vector<int>& nums, int k) {

int l = nums.size();

map<int,int> m;

for(auto i=0;i<l;i++) {

if(!m.count(nums[i])) // if element is not present in m

m[nums[i]] = i;

else {

if(abs(m[nums[i]]-i)<=k) // if the diff is at most k

return true;

else

m[nums[i]] =i; } }

return false;

}

**Report:**

In our map, we will use array elements as key and their index as value. Whenever an element is repeated, we just have to check whether the difference between previous value(using current element as key) and the current index is at most k.

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

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.

**Example:**

int[] nums = new int[] {1,2,3,3,3};

Solution solution = new Solution(nums);

// pick(3) should return either index 2, 3, or 4 randomly. Each index should have equal probability of returning.

solution.pick(3);

// pick(1) should return 0. Since in the array only nums[0] is equal to 1.

solution.pick(1);

**Code:**

class Solution {

public:

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

Solution(vector<int>& nums) {

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

m[nums[i]].push\_back(i);

}

}

int pick(int target) {

return m[target][rand() % m[target].size()];

}

};

**Report:**

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

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

**Example 1:**

**Input:**nums = [1,1,1], k = 2

**Output:** 2

**Code:**

Class Solution {

public:

int subarraySum(vector<int>& nums, int k) {

int count = 0;

unordered\_map<int, int>m;

int sum = 0;

for(auto x: nums){

m[sum]++;

sum += x;

if(m.count(sum - k) > 0) count += m[sum - k];

}

return count;

}

};

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

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.

**Example:**

**Input:**

A = [ 1, 2]

B = [-2,-1]

C = [-1, 2]

D = [ 0, 2]

**Output:**

2

**Code:**

class Solution {

public:

int fourSumCount(vector<int>& A, vector<int>& B, vector<int>& C, vector<int>& D) {

unordered\_map<int, int> absum;

auto cnt = 0;

for (auto a: A)

for (auto b: B)

absum[a + b]++;

for (auto c: C)

for (auto d: D)

cnt += absum[0 - c - d];

return cnt;

}

};

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

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.

**Example:**

**Input:** 19

**Output:** true

**Explanation:**

12 + 92 = 82

82 + 22 = 68

62 + 82 = 100

12 + 02 + 02 = 1

**Code:**

class Solution {

public:

bool isHappy(int n) {

if(n == 0) {

return false;

}

vector<int> seen\_before;

int num = 0;

int num\_sq = 0;

vector<int> digits;

int answer = 0;

while(answer != 1)

{

answer = 0;

while(n>0)

{

num = n%10;

num\_sq = num \* num;

digits.push\_back(num\_sq);

n = n/10;

}

for(int i = 0; i < digits.size(); i++)

{

answer = answer + digits[i];

}

for(int i = 0; i < seen\_before.size(); i++)

{

if(seen\_before[i] == answer)

{

return false;

}

}

seen\_before.push\_back(answer);

digits.clear();

n = answer;

}

return true;

}

};

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

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

**Example 1:**

**Input:** nums = [1,1,1,2,2,3], k = 2

**Output:** [1,2]

**Code:**

Class Solution {

public:

vector<int> topKFrequent(vector<int>& nums, int k) {

vector<int> result;

unordered\_map<int,int> hm;

int maxFreq = 0;

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

maxFreq = max(maxFreq, ++hm[nums[i]]);

vector<vector<int>> freq(maxFreq+1);

for(auto x : hm) freq[x.second].push\_back(x.first);

for(int i = freq.size()-1; i >= 0; i--) {

for(auto x :freq[i]) {

result.push\_back(x);

if(--k == 0) return result;

}

}

return result;

}

};

**Report:**