**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:

bool containsDuplicate(vector<int>& nums) {

unordered\_set<int> set;

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

if ( set.find(nums[i])!=set.end() ){

return true;

}

else {

set.insert(nums[i]);

}

}

return false;

}

};

To solve this problem, we use set to check if set contains nums[i] (the previous element) and if there is such number that equals nums[i] in a set, we return true. Otherwise, we add this element to set.

And if true is not returned, we return false at the end.

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

In this problem, we use map to assign the nums[i] as key and i as their value. For example vector {10, 5, 1}, in map it will be like

class Solution {

public:

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

unordered\_map <int, int> mp;

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

int x = target - nums[j];

if (mp.find(x) != mp.end()){

int i = mp[x];

vector < int > ans{i,j};

return ans;

} else {

mp[nums[j]] = j;

} }

cout<< mp[0];

vector <int> ans;

return ans;

}};

{10:0,5:1,1:2}. Then we found the needed element x = target – nums[j]  
and if there is such element, we get their indexes and store them in “ans” vector.

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

class Solution {

public:

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

unordered\_set < int > a,b;

vector < int > ans;

for (int x : nums1) {

a.insert(x);

}

for (int x : nums2) {

b.insert(x);

}

for (int x : a) {

if (b.find(x) != b.end()){

ans.push\_back(x);

}

}

return ans;

}

};

In this task, we filled two sets a and b with elements of nums1 and nums2 respectively. Then, we just checked if the element x of set ‘a’ is found on set ‘b’, we push this number x to the vector ans. And return the answer at the end.

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

class Solution {

public:

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

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

for (auto x: strs){

string key = x;

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

d[key].push\_back(x); }

vector < vector < string > > ans;

for (auto x: d) {

string key = x.first;

vector <string> value = x.second;

ans.push\_back(value);

} return ans;

}};

First, we filled the map ‘d’. To do this, we sorted the string (as char) and assigned it as key of our map. For example, key of words such bat, tab, bta or atb will be abt. And all this elements will be inserted as value for key. Next, we should insert these to the answer vector. To do this, we just loop through the values of ‘d’- and push them as vector to answer vector which contains vector of vectors.

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

class Solution {

public:

bool wordPattern(string pattern, string s) {

unordered\_map <string, char> da;

unordered\_map <char, string> db;

vector < string > words;

string cur\_word = "";

for (auto x: s){

if (x == ' '){

words.push\_back(cur\_word);

cur\_word = "";

} else {

cur\_word += x;

}

}

words.push\_back(cur\_word);

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

return false;

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

auto ch = pattern[i];

auto wr = words[i];

bool case1 = (da.find(wr) != da.end() && da[wr] == ch);

bool case2 = (db.find(ch) != db.end() && db[ch] == wr);

if (case1 && case2) {

continue;

}

else {

if (da.find(wr) != da.end() || db.find(ch) != db.end()){

return false;

}

else {

da[wr] = ch;

db[ch] = wr;

}

}

}

return true;

}

};

To solve this, we used double binding. Firstly we separated the words in string s to vector words.

And then we created 2 bool cases where we check if the word is in our ‘da’ map (stores words as keys) and ‘db’ stores chars as words. In 2 bool cases we check if they are met equally, da[wr] == ch and db[ch] == wr and if it is so we continue. If only one condition is met we return false. Else we add them to maps as

da[wr] = ch;

db[ch] = wr;

and return true.

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

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

Here we store the elements of vector nums as keys and their positions are values.

First, we check if there is already such key nums[i] in our map and if the difference between i and the position of this (key nums[i]) is less or equal to k, we return true. Otherwise, we add to map like nums[i] = key and i = position. At the end, we return false if there is no nearby duplicates are found.

class Solution {

public:

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

unordered\_map <int, int> d;

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

if (d.find(nums[i]) != d.end() && abs(i - d.find(nums[i])->second )<= k){

return true;

}

else {

d[nums[i]] = i;

}

}

return false;

}

};

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

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

class Solution {

public:

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

vector < int > prefixSum(nums.size(), 0);

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

prefixSum[i] += nums[i];

if (i > 0) {

prefixSum[i] += prefixSum[i - 1];

}

}

unordered\_map < int, int > d;

int ans = 0;

d[0]++;

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

int x = prefixSum[i] - k;

if (d.find(x) != d.end()){

ans += d[x];

}

d[prefixSum[i]]++;

}

return ans;

}

};

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

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

class Solution {

public:

unordered\_set <int> set;

int rep(int n) {

int sum=0;

while (n > 0){

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

n = n/10;

}

cout<<sum<<endl;

return sum;

}

bool isHappy(int n) {

if (this->set.find(n) != this->set.end())

return false;

else if (rep(n) == 1){

return true;

}

else

this->set.insert(n);

return isHappy(rep(n));

return true;

}

};

To solve this problem, I declared global set and created method rep that gets sum of square root digits of number n. And in bool method, I checked if there is such element ‘n’ in our set and if it is, we return false, and else if the rep(n) equals 1, we return true. Otherwise, we add this number to our set and return the rep(n). So this works recursively, for example:

4 = 16

16 = 1 + 36

37 = 9 + 49

58 = 25 + 64

89 = 64 + 81

145 = 1 + 16 +25

42 = 16 + 4

20 = 4

And here we have {4,16,37,58,89,145,45,20} in our set and if me meet the 4 again, the false will be returned.

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