**Probleme pentru activitate seminar**

(Username cont Leetcod: micul)

**1. Find the Difference**

class Solution {

public:

char findTheDifference(string s, string t) {

char diff = 0;

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

diff = diff ^ s[i];

}

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

diff = diff ^ t[i];

}

return diff;

}

};

**2. 4Sum II**

class Solution {

public:

int fourSumCount(vector<int>& nums1, vector<int>& nums2, vector<int>& nums3, vector<int>& nums4) {

int n = nums1.size(), res = 0;

unordered\_map<int, int> map;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

map[nums1[i] + nums2[j]]++;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

res += map[-nums3[i] - nums4[j]];

return res;

}

};

**3. Substring with Concatenation of All Words**

class Solution {

public:

vector<int> findSubstring(string s, vector<string>& words) {

unordered\_map<string, int> map;

unordered\_map<string, int> copy;

vector<int> res;

int len\_s = s.length(), nr\_words = words.size(), len\_word = words[0].length(), j;

for (int i = 0; i < nr\_words; i++){

map[words[i]]++;

}

for (int i = 0; i <= len\_s - len\_word \* nr\_words; i++){

copy = map;

for(j = i; j < i + nr\_words \* len\_word; j += len\_word){

string word = s.substr(j, len\_word);

if (copy[word])

copy[word]--;

else

break;

}

if (j - i == nr\_words \* len\_word)

res.push\_back(i);

}

return res;

}

};

**Tema seminar 4**

**202. Happy Number (https://leetcode.com/problems/happy-number/)**

**Cerinta**: Un numar este fericit daca prin insumarea repetata a patratelor cifrelor sale se obtine 1. Sa se scrie un algoritm care determina daca un numar dat e fericit sau nu.

**Rezolvare:**

Prin insumarea repetata a patratelor cifrelor unui numar se pot distinge trei cazuri:

1. se ajunge la 1 si atunci numarul este fericit

2. se ajunge la un ciclu care nu il contine pe 1 si atunci numarul nu este fericit

3. suma creste spre infinit, numarul nu e fericit

Al treilea caz de fapt nu este posibil in problema deoarece daca analizam cel mai mare numar pe care il putem obtine prin insumarea patratelor cifrelor vom obtine un numar finit. Cea mai mare valoare pentru n este 2^31 - 1 =2147483647 care are 10 cifre. Cum cel mai mare numar de 10 cifre e 9999999999 care genereaza si cea mai mare suma a patratelor cifrelor si anume 9^2+9^2+9^2.... = 810 inseamna ca din cele 10 cifre se ajung la maxim 3, deci suma cifrelor nu are cum sa tinda spre infinit.

Asta mai inseamna si ca ori se ajunge la un moment dat la 1 ori exista un ciclu care mentine suma patratelor cifrelor sub acea valoare finita. Deci de fapt in problema sunt doar doua cazuri de tratat.

Un ciclu se poate detecta in momentul in care am obtinut un numar intermediar pe care l-am mai obtinut inainte in incercarea de a determina daca numarul dat este fericit sau nu. Pentru asta ne putem folosi de un unordered\_set<int> in care sa adaugam numerele intermediare pe masura ce la generam si asa putem verifica daca un numar a mai fost obtinut sau nu.

Pentru rezolvarea problemei vom avea nevoie de o functie care sa calculeze urmatorul numar si de fiecare data trebuie facuta verificarea daca a aparut un ciclu sau s-a obtinut numarul 1. Functia care returneaza urmatorul numar calculeaza suma cifrelor din numarul curent ridicate la patrat. Parcurgerea numarului se face de la final la inceput, eliminand la fiecare pas cate o cifra pana cand le-am adaugat pe toate la suma:

int getNext(int n) {

int sum = 0;

while (n > 0){

int digit = n % 10;

n = n / 10;

sum += digit \* digit;

}

return sum;

}

Verificam dupa fiecare generare daca numarul acela a mai fost obtinut (se afla in set), caz in care s-a obtinut un ciclu sau este 1, caz in care numarul este fericit.

bool isHappy(int n) {

unordered\_set<int> set;

while (n != 1 && set.find(n) == set.end()){ // continuam cat timp numarul

// generat nu e 1 sau nu a mai

// aparut in set (nu am gasit un

// ciclu)

set.insert(n); // adaugam numarul curent in set

n = getNext(n); // generam urmatorul numar

}

if (n == 1) // daca am ajuns la 1 inseamna ca numarul e fericit

return true;

else // altfel am dat de un ciclu iar numarul nu e fericit

return false;

}

Complexitatea timp a functiei de generare a urmatorului numar este O(logn) deoarece sunt parcurse toate cifrele numarului n si facuta o operatie de adunare pe ele, iar numarul de cifre al unui numar este logn. Complexitatea de cautare intr-un unordered\_set este in medie O(1).

Complexitatea timp totala se calculeaza in functie de cate numere vor fi generate pana se decide daca este un numar fericit sau nu si cat de mari sunt aceste numere.

Pentru numere de trei cifre sau mai mici, suma maxima a patratelor cifrelor este 243 pentru ca numarul maxim de trei cifre este 999 care genereaza si suma maxima (9^2+9^2+9^2 = 243). Astfel, orice numar de trei cifre sau mai mic nu va putea sa genereze un numar mai mare de 243 pentru ca suma maxima a cifrelor numarului este 243. Din acest moment programul poate sa cicleze in cel mai rau caz prin maxim 243 de numere pana ajunge la 1 sau se repeta un numar. In cazul in care numarul initial are cel mult 10 cifre (pentru n maxim) numarul maxim care poate fi generat este 810 (explicat mai sus), care are 3 cifre deci se va ajunge tot in cazul descris mai sus in care dupa 243 se va ajunge ori la 1 ori la un ciclu. Fiecare dintre numerele care vor fi generate pe urma au cel mult 3 cifre, deci pentru a calcula suma cifrelor tuturor algoritmul executa maxim 243 \* 3 pasi. Astfel, complexitate de timp a algoritmului este O(243 \* 3 + logn + loglogn + logloglogn + ...) = O(logn).

Complexitatea de spatiu este O(logn).

**141. Linked List Cycle (**[**https://leetcode.com/problems/linked-list-cycle/**](https://leetcode.com/problems/linked-list-cycle/)**)**

**Cerinta:** Se primeste un pointer la nodul de inceput al unei liste inlantuite si trebuie sa se determine daca lista are un ciclu sau nu.

**Rezolvare:**

Se parcurge lista inlantuita de la inceput spre final si se adauga pe rand fiecare nod intr-un unordered\_set pentru a putea vedea daca acelasi nod va aparea de doua ori. Daca se intampla asta insemna ca acolo incepe un ciclu.

bool hasCycle(ListNode \*head) {

unordered\_set<ListNode\*> set;

while(head != NULL){

if(set.find(head) == set.end()){

set.insert(head);

} else{

return true;

}

head = head->next;

}

return false;

}

Complexitatea de timp este in medie O(n), deoarece se face o singura parcurgere, iar cautarea in set se face in medie in O(1), dar in cel mai rau caz in O(n), care determina o complexitate totala de O(n^2), unde n e numarul de noduri din lista inlantuita.

Complexitatea de spatiu este O(2n)=O(n), pentru ca avem n noduri in lista si inca n in set.