Hashing, sliding window, prefixsum
Professor Wladimir

Hashing

Em programação competitiva, a técnica de hashing consiste em associar valores a chaves únicas (ou quase únicas), permitindo o acesso eficiente a esses valores, geralmente em tempo constante na média. Na linguagem C++, utilizamos as seguintes estruturas de dados:

- unordered_set
- unordered_map

Problemas clássicos para utilização do hashing:

- detecção de duplicadas
- contagem de frequência
- verificar anagramas
- detecção de ciclos em jogos/tabuleiros
- encontrar subvetores com soma k
- encontrar dois números com soma igual a zero.

217. Contains Duplicate

url: https://leetcode.com/problems/contains-duplicate/
Neste problema, associaremos a cada número presente no vetor nums a quantidade de vezes que ele ocorre nesse mesmo vetor.

```
bool containsDuplicate(vector<int>& nums) {
       unordered_map <int, int> freq;

       for(int x : nums){
           if( freq.count(x) == 1) return true;
           else freq[x]++;
       }

       return false;
   }
```

448. Find All Numbers Disappeared in an Array

url: https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/description/

```cpp
vector<int> findDisappearedNumbers(vector<int>& nums) {
    vector <int> h;
    vector <int> res;
    h.assign( nums.size(), 0);

    for(int x : nums) h[x-1]++;

    for(int i = 0; i < nums.size(); i++){
        if( h[i] == 0)
            res.push_back(i+1);
    }

    return res;

}
```

Valid Anagram

URL: https://neetcode.io/problems/is-anagram

```cpp
bool isAnagram(string s, string t) {
    unordered_map <char, int> freq;

    for(char c : s){
        freq[c]++;
    }

    for(auto c : t){
        freq[c]--;
    }
    typeid(freq)::iterator it;

    for ( it = freq.begin(); it != freq.end(); ++it){
        if( it->second != 0) return false;
    }

    return true;
}
```

```cpp
bool isAnagram(string s, string t) {
    unordered_map <char, int> freq;

    for(char c : s){
        freq[c]++;
    }

    for(auto c : t){
        freq[c]--;
    }

    for ( auto [k,v] : freq){
        if( v != 0) return false;
    }

    return true;
}
```

202. Happy Number

url: https://leetcode.com/problems/happy-number/description/

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.
- 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.

Return true if n is a happy number, and false if not.

```
bool isHappy(int n) {
    unordered_set <int> s;
    while( n!= 1 && s.count(n) == 0){
        int t = n;
        int soma = 0;
        //cout << " n " << n << endl;
        while(t != 0){
            int d = t%10;
            soma += d*d;
            t = t/10;
        }
        //cout << "s " << soma << endl;
        s.insert(n);
        n = soma;
    }
    if(n==1) return true;
    else return false;
}
```

Hashing + prefixsum

560. Subarray Sum Equals K

url : https://leetcode.com/problems/subarray-sum-equals-k/description/

Given an array of integers nums and an integer k, return the total number of subarrays whose sum equals to k.

A subarray is a contiguous non-empty sequence of elements within an array.

Example 1:

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

Example 2:

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

A ideia central do algoritmo é utilizar uma abordagem baseada em somas acumuladas para contar, de forma eficiente, quantos subvetores consecutivos possuem soma exatamente igual a k.

Para isso, o algoritmo mantém um mapa de frequências que registra quantas vezes cada valor de soma acumulada já ocorreu até determinado ponto do vetor. Inicialmente, assumimos que a soma acumulada 0 ocorreu uma vez — isso representa a situação antes de qualquer elemento ser processado, o que permite identificar subvetores que começam na posição inicial do vetor.

Durante a execução, a cada novo elemento processado, atualizamos a soma acumulada atual acc. Para verificar se existe algum subvetor terminado na posição atual cuja soma seja k, basta verificar se a diferença $acck$ já foi registrada anteriormente no mapa. Se essa diferença já apareceu l vezes, isso indica que existem l subvetores com soma exatamente igual a k terminando na posição atual.

Esses subvetores correspondem a todos os trechos do vetor cuja soma acumulada no início era $acck$ e que, ao chegar em acc, acumularam exatamente k. Por fim, o mapa é atualizado para refletir que o valor acc também foi alcançado.

```cpp
int subarraySum(vector<int>& nums, int k) {
    int count = 0;
    map<int, int> mapa;
    int n = nums.size();
    int acc = 0;

    mapa[0] = 1;
    for(int i = 0; i < n; i++){
        acc += nums[i];
        if(  mapa.count( acc - k ) ){
            count += mapa[acc - k] ;
        }
        mapa[acc]++;
    }
    return count;
}
```

974. Subarray Sums Divisible by K

Given an integer array nums and an integer k, return the number of non-empty subarrays that have a sum divisible by k.

A subarray is a contiguous part of an array.

Example 1:

Input: nums = [4,5,0,-2,-3,1], k = 5
Output: 7
Explanation: There are 7 subarrays with a sum divisible by k = 5: [4, 5, 0, -2, -3, 1], [5], [5, 0], [5, 0, -2, -3], [0], [0, -2, -3], [-2, -3]

Example 2:

Input: nums = [5], k = 9
Output: 0

A ideia é usar prefixos acumulados e propriedades de congruência modular:

- Suponha que a soma dos primeiros i elementos seja $acc[i]$.
- Se $acc[j] - acc[i]$ é divisível por k, então o subarray $nums[i+1..j]$ tem soma divisível por k.

Ou seja, se dois prefixos tiverem o mesmo valor módulo k, então a diferença entre eles é múltiplo de k.

```cpp
int modulo(int a, int b) {
    int r = a % b;
    return (r < 0) ? r + b : r;
}

int subarraysDivByK(vector<int>& nums, int k) {
    unordered_map <int, int> freq;
    int acc = 0;
    int count = 0;
    freq[0] = 1;
    for(int x : nums){
        acc += x ;
        acc = modulo(acc, k);
        count += freq[acc];
        freq[acc]++;

    }
    return count;
}
```

Exercícios Sugeridos

- 1941. Check if All Characters Have Equal Number of Occurrences https://leetcode.com/problems/check-if-all-characters-have-equal-number-of-occurrences/description/
- 2006. Count Number of Pairs With Absolute Difference K https://leetcode.com/problems/count-number-of-pairs-with-absolute-difference-k/description/
- 219. Contains Duplicate II https://leetcode.com/problems/contains-duplicate-ii/description/
- 242. Valid Anagram https://leetcode.com/problems/valid-anagram/description/