LeetCode Training Day 1 Hashtable

Hash table is a common data structure which is used to store key value pair. The access to an hash table element is in O(1) time complex. The hash table is normally used for storing the mapping value or the element count. Since the duplicated key in hash table will only have one storage space, so hash table sometimes is used for de-dup.

Please notice to access hashtable, you need to first check if the key is in the hashtable.

In some special case, we only want to store the distinct value but no count or mapping values, then hash table can be reduced to a set.

Please notice that in many languages, there is another data structure which stored the key in order, this is not the topic today, will cover later.

The following is the hashtable definition in common languages.

|  |  |  |
| --- | --- | --- |
| Language | Hashtable | Set |
| C++ | unordered\_map<T, T> | unordered\_set<T> |
| C# | Dictionary<T, T> | HashSet<T> |
| Java | Hashtable<T, T> | HashSet<T> |

The common operations are here:

|  |  |
| --- | --- |
| Language | Operations: |
| C++ | Check if key exist: if (hashtable.count(key) > 0)  Set value: hashtable[key] = value;  Get value: value = hashtable[key];  Get size of table: hashtable.size();  Erase a key: hashtable.erase(key);  Iteration: for (auto itr : hashtable) {key = itr.first; value = itr.second;} |
| C# | Check if key exist: if (hashtable.ContainsKey(key))  Set value: hashtable.Add(key, value);  Get value: value = hashtable[key];  Get size of table: hashtable.Count;  Erase a key: hashtable.Remove(key);  Iteration: foreach( KeyValuePair<T, T> kvp in hashtable)  { key = kvp.first; value = kvp.second;};  foreach(T key in hashtable)  { value = hashtable[key];} |
| Java | Check if key exist: if (hashtable.ContainsKey(key))  Set value: hashtable.put(key, value);  Get value: value = hashtable.get(key);  Get size of table: hashtable.size();  Erase a key: hashtable.remove(key);  Iteration: Set<Entry<T, T>> entrySet = hashtable.entrySet();  for (Entry<T, T> entry: entrySet)  { key = entry.getKey(); value = entry.getKey();}; |

# Anagram

Anagram means two words has the exactly same characters.

## 242. Valid Anagram

Easy

Given two strings s and t, return true *if* t *is an anagram of* s*, and* false *otherwise*.

**Example 1:**

**Input:** s = "anagram", t = "nagaram"

**Output:** true

**Example 2:**

**Input:** s = "rat", t = "car"

**Output:** false

**Constraints:**

* 1 <= s.length, t.length <= 5 \* 104
* s and t consist of lowercase English letters.

### Analysis:

we use hash table to count the characters for string one, then deduct the count for string 2, if all count become zero then it is anagram.

Please note that the following solution is not unique for such problem, you can sort the two strings and compare them, the code is even shorter or you can use vector of 26 which represent ‘a’ to ‘z’ since all are lower cases.

/// <summary>

/// Leet code #242. Valid Anagram

///

/// Given two strings s and t, write a function to determine if t is an

/// anagram of s.

/// For example,

/// s = "anagram", t = "nagaram", return true.

/// s = "rat", t = "car", return false.

/// Note:

/// You may assume the string contains only lowercase alphabets.

/// Follow up:

/// What if the inputs contain unicode characters? How would you adapt

/// your solution to such case?

/// </summary>

bool LeetCodeString::isAnagram(string s, string t)

{

if (s.size() != t.size()) return false;

unordered\_map<char, int> map;

for (size\_t i = 0; i < s.size(); i++)

{

map[s[i]]++;

}

for (size\_t i = 0; i < t.size(); i++)

{

map[t[i]]--;

if (map[t[i]] == 0) map.erase(t[i]);

}

if (map.size() == 0) return true;

else return false;

}

## 205. Isomorphic Strings

Easy

Given two strings s and t, *determine if they are isomorphic*.

Two strings s and t are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

**Example 1:**

**Input:** s = "egg", t = "add"

**Output:** true

**Example 2:**

**Input:** s = "foo", t = "bar"

**Output:** false

**Example 3:**

**Input:** s = "paper", t = "title"

**Output:** true

**Constraints:**

* 1 <= s.length <= 5 \* 104
* t.length == s.length
* s and t consist of any valid ascii character.

### Analysis:

Map the character from source to destination, and check only 1:1 mapping

/// <summary>

/// Leet code #205. Isomorphic Strings

///

/// Given two strings s and t, determine if they are isomorphic.

/// Two strings are isomorphic if the characters in s can be replaced to

/// get t.

/// All occurrences of a character must be replaced with another character

/// while preserving the order of characters. No two characters may map to

/// the same character but a character may map to itself.

/// For example,

/// Given "egg", "add", return true.

///

/// Given "foo", "bar", return false.

/// Given "paper", "title", return true.

/// Note:

/// You may assume both s and t have the same length.

/// </summary>

bool LeetCodeString::isIsomorphic(string s, string t)

{

unordered\_map<char, char> src\_map, dst\_map;

for (size\_t i = 0; i < s.size(); i++)

{

if (src\_map.count(s[i]) == 0 && dst\_map.count(t[i]) == 0)

{

src\_map[s[i]] = t[i];

dst\_map[t[i]] = s[i];

}

else if (src\_map[s[i]] != t[i] || dst\_map[t[i]] != s[i])

{

return false;

}

}

return true;

}

## 202. Happy Number

Easy

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*.

**Example 1:**

**Input:** n = 19

**Output:** true

**Explanation:**

12 + 92 = 82

82 + 22 = 68

62 + 82 = 100

12 + 02 + 02 = 1

**Example 2:**

**Input:** n = 2

**Output:** false

**Constraints:**

* 1 <= n <= 231 - 1

### Analysis:

Use set to check loop.

/// <summary>

/// Leet code #202. Happy Number

/// Write an algorithm to determine if a number 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.

/// Example: 19 is a happy number

/// 1^2 + 9^2 = 82

/// 8^2 + 2^2 = 68

/// 6^2 + 8^2 = 100

/// 1^2 + 0^2 + 0^2 = 1

/// </summary>

bool LeetCodeHashtable::isHappy(int n)

{

unordered\_set<int> data\_set;

while (true)

{

data\_set.insert(n);

int sum = 0;

while (n != 0)

{

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

n = n / 10;

}

n = sum;

if ((n == 1) || (data\_set.find(n) != data\_set.end()))

{

break;

}

}

if (n == 1) return true;

else return false;

}

## 1. Two Sum

Easy

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

You may assume that each input would have ***exactly*** one solution, and you may not use the *same* element twice.

**Example:**

Given nums = [2, 7, 11, 15], target = 9,

Because nums[**0**] + nums[**1**] = 2 + 7 = 9,

return [**0**, **1**].

### Analysis:

we can store the previously visited number in the hash table, and for any number we check if target – num[i] is in the hash table.

/// <summary>

/// Leet code 1 Two Sum

/// Given an array of integers, return indices of the two numbers such that

/// they add up to a specific target.

/// You may assume that each input would have exactly one solution.

/// Example:

/// Given nums = [2, 7, 11, 15], target = 9,

/// Because nums[0] + nums[1] = 2 + 7 = 9,

/// return[0, 1]

/// </summary>

vector<int> LeetCode::twoSum(vector<int>& nums, int target)

{

vector<int> result;

unordered\_map<int, int> num\_map;

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

{

if (num\_map.count(target - nums[i]) > 0)

{

result = { num\_map[target - nums[i]], i };

return result;

}

num\_map[nums[i]] = i;

}

return result;

}

Similar Problems:

https://leetcode.com/problems/max-number-of-k-sum-pairs/

## 2032. Two Out of Three

Easy

Given three integer arrays nums1, nums2, and nums3, return *a****distinct****array containing all the values that are present in****at least two****out of the three arrays. You may return the values in****any****order*.

**Example 1:**

**Input:** nums1 = [1,1,3,2], nums2 = [2,3], nums3 = [3]

**Output:** [3,2]

**Explanation:** The values that are present in at least two arrays are:

- 3, in all three arrays.

- 2, in nums1 and nums2.

**Example 2:**

**Input:** nums1 = [3,1], nums2 = [2,3], nums3 = [1,2]

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

**Explanation:** The values that are present in at least two arrays are:

- 2, in nums2 and nums3.

- 3, in nums1 and nums2.

- 1, in nums1 and nums3.

**Example 3:**

**Input:** nums1 = [1,2,2], nums2 = [4,3,3], nums3 = [5]

**Output:** []

**Explanation:** No value is present in at least two arrays.

**Constraints:**

* 1 <= nums1.length, nums2.length, nums3.length <= 100
* 1 <= nums1[i], nums2[j], nums3[k] <= 100

### Analysis:

We use value of numbers as key and array id in a set as value, please notice that hashtable value can be complex type.

/// <summary>

/// Leet code 2032. Two Out of Three

///

/// Easy

///

/// Given three integer arrays nums1, nums2, and nums3, return a distinct

/// array containing all the values that are present in at least two out

/// of the three arrays. You may return the values in any order.

///

/// Example 1:

/// Input: nums1 = [1,1,3,2], nums2 = [2,3], nums3 = [3]

/// Output: [3,2]

/// Explanation: The values that are present in at least two arrays are:

/// - 3, in all three arrays.

/// - 2, in nums1 and nums2.

///

/// Example 2:

/// Input: nums1 = [3,1], nums2 = [2,3], nums3 = [1,2]

/// Output: [2,3,1]

/// Explanation: The values that are present in at least two arrays are:

/// - 2, in nums2 and nums3.

/// - 3, in nums1 and nums2.

/// - 1, in nums1 and nums3.

///

/// Example 3:

/// Input: nums1 = [1,2,2], nums2 = [4,3,3], nums3 = [5]

/// Output: []

/// Explanation: No value is present in at least two arrays.

///

/// Constraints:

/// 1. 1 <= nums1.length, nums2.length, nums3.length <= 100

/// 2. 1 <= nums1[i], nums2[j], nums3[k] <= 100

/// </summary>

vector<int> LeetCodeHashtable::twoOutOfThree(vector<int>& nums1, vector<int>& nums2, vector<int>& nums3)

{

vector<set<int>> set\_map(101);

vector<int> result;

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

{

set\_map[nums1[i]].insert(1);

}

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

{

set\_map[nums2[i]].insert(2);

}

for (size\_t i = 0; i < nums3.size(); i++)

{

set\_map[nums3[i]].insert(3);

}

for (size\_t i = 0; i < set\_map.size(); i++)

{

if (set\_map[i].size() >= 2) result.push\_back(i);

}

return result;

}

## 219. Contains Duplicate II

Easy

Given an integer array nums and an integer k, return true if there are two **distinct indices** i and j in the array such that nums[i] == nums[j] and abs(i - j) <= k.

**Example 1:**

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

**Output:** true

**Example 2:**

**Input:** nums = [1,0,1,1], k = 1

**Output:** true

**Example 3:**

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

**Output:** false

**Constraints:**

* 1 <= nums.length <= 105
* -109 <= nums[i] <= 109
* 0 <= k <= 105

### Analysis:

We track the position of number in a hashtable.

/// <summary>

/// Leet code #219. 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 difference between i and j is at most k.

/// </summary>

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

{

unordered\_map<int, int> pos\_map;

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

{

if (pos\_map.find(nums[i]) != pos\_map.end())

{

if ((int)(i - pos\_map[nums[i]]) <= k) return true;

}

pos\_map[nums[i]] = i;

}

return false;

}

## 347. Top K Frequent Elements

Medium

Given an integer array nums and an integer k, return *the* k *most frequent elements*. You may return the answer in **any order**.

**Example 1:**

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

**Output:** [1,2]

**Example 2:**

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

**Output:** [1]

**Constraints:**

* 1 <= nums.length <= 105
* k is in the range [1, the number of unique elements in the array].
* It is **guaranteed** that the answer is **unique**.

### Analysis:

we use hash table to count the number first, the put the top K into priority queue (use negative to make smallest on top)

/// <summary>

/// Leet code #347. Top K Frequent Elements

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

/// For example,

/// Given [1,1,1,2,2,3] and k = 2, return [1,2].

/// Note:

/// You may assume k is always valid, 1 <= k <= number of unique elements.

/// Your algorithm's time complexity must be better than O(n log n),

/// where n is the array's size.

/// </summary>

vector<int> LeetCodeHashtable::topKFrequent(vector<int>& nums, int k)

{

vector<int> result;

unordered\_map<int, int> num\_count;

for (int num : nums) num\_count[num]++;

priority\_queue<pair<int, int>> pq;

for (auto itr : num\_count)

{

pair<int, int> pair = make\_pair(-itr.second, itr.first);

pq.push(pair);

if (pq.size() > (size\_t)k) pq.pop();

}

while (!pq.empty())

{

result.push\_back(pq.top().second);

pq.pop();

}

std::reverse(result.begin(), result.end());

return result;

}

Similar Problem:

<https://leetcode.com/problems/most-common-word/>

<https://leetcode.com/problems/least-number-of-unique-integers-after-k-removals/>

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

# Advanced Problems

## 1814. Count Nice Pairs in an Array

Medium

You are given an array nums that consists of non-negative integers. Let us define rev(x) as the reverse of the non-negative integer x. For example, rev(123) = 321, and rev(120) = 21. A pair of indices (i, j) is **nice** if it satisfies all of the following conditions:

* 0 <= i < j < nums.length
* nums[i] + rev(nums[j]) == nums[j] + rev(nums[i])

Return *the number of nice pairs of indices*. Since that number can be too large, return it **modulo** 109 + 7.

**Example 1:**

**Input:** nums = [42,11,1,97]

**Output:** 2

**Explanation:** The two pairs are:

- (0,3) : 42 + rev(97) = 42 + 79 = 121, 97 + rev(42) = 97 + 24 = 121.

- (1,2) : 11 + rev(1) = 11 + 1 = 12, 1 + rev(11) = 1 + 11 = 12.

**Example 2:**

**Input:** nums = [13,10,35,24,76]

**Output:** 4

**Constraints:**

* 1 <= nums.length <= 105
* 0 <= nums[i] <= 109

### Analysis:

If we change the formula: nums[i] + rev(nums[j]) == nums[j] + rev(nums[i]) to

nums[i] - rev(nums[i]) == nums[j] - rev(nums[j]), then each element can calculate its value without referencing others.

We need some string conversion to convert integer with string

/// <summary>

/// Leet code 1814. Count Nice Pairs in an Array

///

/// Medium

///

/// You are given an array nums that consists of non-negative integers.

/// Let us define rev(x) as the reverse of the non-negative integer x.

/// For example, rev(123) = 321, and rev(120) = 21. A pair of

/// indices (i, j) is nice if it satisfies all of the following conditions:

///

/// 0 <= i < j < nums.length

/// nums[i] + rev(nums[j]) == nums[j] + rev(nums[i])

/// Return the number of nice pairs of indices. Since that number can be

/// too large, return it modulo 10^9 + 7.

///

/// Example 1:

/// Input: nums = [42,11,1,97]

/// Output: 2

/// Explanation: The two pairs are:

/// - (0,3) : 42 + rev(97) = 42 + 79 = 121, 97 + rev(42) = 97 + 24 = 121.

/// - (1,2) : 11 + rev(1) = 11 + 1 = 12, 1 + rev(11) = 1 + 11 = 12.

///

/// Example 2:

/// Input: nums = [13,10,35,24,76]

/// Output: 4

///

/// Constraints:

/// 1. 1 <= nums.length <= 10^5

/// 2. 0 <= nums[i] <= 10^9

/// </summary>

int LeetCodeHashtable::countNicePairs(vector<int>& nums)

{

long long M = 1000000007;

unordered\_map<long long, long long> num\_count;

long long result = 0;

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

{

string str = to\_string(nums[i]);

reverse(str.begin(), str.end());

long long rev = atol(str.c\_str());

long long value = (long long)(nums[i]) - rev;

result = (result + num\_count[value]) % M;

num\_count[value]++;

}

return (int)result;

}

Similar Problem

<https://leetcode.com/problems/number-of-good-pairs/>

## 1452. People Whose List of Favorite Companies Is Not a Subset of Another List

Medium

Given the array favoriteCompanies where favoriteCompanies[i] is the list of favorites companies for the ith person (**indexed from 0**).

*Return the indices of people whose list of favorite companies is not a****subset****of any other list of favorites companies*. You must return the indices in increasing order.

**Example 1:**

**Input:** favoriteCompanies = [["leetcode","google","facebook"],["google","microsoft"],["google","facebook"],["google"],["amazon"]]

**Output:** [0,1,4]

**Explanation:**

Person with index=2 has favoriteCompanies[2]=["google","facebook"] which is a subset of favoriteCompanies[0]=["leetcode","google","facebook"] corresponding to the person with index 0.

Person with index=3 has favoriteCompanies[3]=["google"] which is a subset of favoriteCompanies[0]=["leetcode","google","facebook"] and favoriteCompanies[1]=["google","microsoft"].

Other lists of favorite companies are not a subset of another list, therefore, the answer is [0,1,4].

**Example 2:**

**Input:** favoriteCompanies = [["leetcode","google","facebook"],["leetcode","amazon"],["facebook","google"]]

**Output:** [0,1]

**Explanation:** In this case favoriteCompanies[2]=["facebook","google"] is a subset of favoriteCompanies[0]=["leetcode","google","facebook"], therefore, the answer is [0,1].

**Example 3:**

**Input:** favoriteCompanies = [["leetcode"],["google"],["facebook"],["amazon"]]

**Output:** [0,1,2,3]

**Constraints:**

* 1 <= favoriteCompanies.length <= 100
* 1 <= favoriteCompanies[i].length <= 500
* 1 <= favoriteCompanies[i][j].length <= 20
* All strings in favoriteCompanies[i] are **distinct**.
* All lists of favorite companies are **distinct**, that is, If we sort alphabetically each list then favoriteCompanies[i] != favoriteCompanies[j].
* All strings consist of lowercase English letters only.

### Analysis:

We use bit map to indicate a company is loved by ith person, then for a particular person, if besides himself, all the companies he loved is also loved by any at least one person then he is covered, we use AND to check if bit map covers.

/// <summary>

/// Leet code #1452. People Whose List of Favorite Companies Is Not a

/// Subset of Another List

///

/// Medium

///

/// Given the array favoriteCompanies where favoriteCompanies[i] is the

/// list of favorites companies for the ith person (indexed from 0).

/// Return the indices of people whose list of favorite companies is not

/// a subset of any other list of favorites companies. You must return

/// the indices in increasing order.

///

/// Example 1:

/// Input: favoriteCompanies = [["leetcode","google","facebook"],

/// ["google","microsoft"],["google","facebook"],["google"],["amazon"]]

/// Output: [0,1,4]

/// Explanation:

/// Person with index=2 has favoriteCompanies[2]=["google","facebook"]

/// which is a subset of favoriteCompanies[0]=["leetcode","google",

/// "facebook"] corresponding to the person with index 0.

/// Person with index=3 has favoriteCompanies[3]=["google"] which is a

/// subset of favoriteCompanies[0]=["leetcode","google","facebook"] and

/// favoriteCompanies[1]=["google","microsoft"].

/// Other lists of favorite companies are not a subset of another list,

/// therefore, the answer is [0,1,4].

///

/// Example 2:

/// Input: favoriteCompanies = [["leetcode","google","facebook"],

/// ["leetcode","amazon"],["facebook","google"]]

/// Output: [0,1]

/// Explanation: In this case favoriteCompanies[2]=["facebook","google"]

/// is a subset of favoriteCompanies[0]=["leetcode","google","facebook"],

/// therefore, the answer is [0,1].

///

/// Example 3:

/// Input: favoriteCompanies = [["leetcode"],["google"],["facebook"],

/// ["amazon"]]

/// Output: [0,1,2,3]

///

/// Constraints:

/// 1. 1 <= favoriteCompanies.length <= 100

/// 2. 1 <= favoriteCompanies[i].length <= 500

/// 3. 1 <= favoriteCompanies[i][j].length <= 20

/// 4. All strings in favoriteCompanies[i] are distinct.

/// 5. All lists of favorite companies are distinct, that is, If we sort

/// alphabetically each list then

/// favoriteCompanies[i] != favoriteCompanies[j].

/// 6. All strings consist of lowercase English letters only.

/// </summary>

vector<int> LeetCodeHashtable::peopleIndexes(vector<vector<string>>& favoriteCompanies)

{

unordered\_map<string, bitset<100>> company\_masks;

vector<int> result;

for (size\_t i = 0; i < favoriteCompanies.size(); i++)

{

for (size\_t j = 0; j < favoriteCompanies[i].size(); j++)

{

string company = favoriteCompanies[i][j];

company\_masks[company].set(i);

}

}

for (size\_t i = 0; i < favoriteCompanies.size(); i++)

{

bitset<100> bit\_mask;

bit\_mask.set();

for (size\_t j = 0; j < favoriteCompanies[i].size(); j++)

{

string company = favoriteCompanies[i][j];

bitset<100> company\_mask = company\_masks[company];

// discard self bit

company\_mask.reset(i);

bit\_mask &= company\_mask;

if (bit\_mask.none()) break;

}

if (bit\_mask.none()) result.push\_back(i);

}

return result;

}

## 128. Longest Consecutive Sequence

Medium

Given an unsorted array of integers nums, return *the length of the longest consecutive elements sequence.*

You must write an algorithm that runs in O(n) time.

**Example 1:**

**Input:** nums = [100,4,200,1,3,2]

**Output:** 4

**Explanation:** The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.

**Example 2:**

**Input:** nums = [0,3,7,2,5,8,4,6,0,1]

**Output:** 9

**Constraints:**

* 0 <= nums.length <= 105
* -109 <= nums[i] <= 109

### Analysis:

We first put all numbers in a hashset, then take a number, iterate the numbers less than it or greater than it, then remove these iterated numbers from hashset.

/// <summary>

/// Leet code #128. Longest Consecutive Sequence

/// Given an unsorted array of integers, find the length of the longest

/// consecutive elements sequence.

/// For example,

/// Given [100, 4, 200, 1, 3, 2],

/// The longest consecutive elements sequence is [1, 2, 3, 4].

/// Return its length: 4.

/// Your algorithm should run in O(n) complexity.

/// </summary>

int LeetCodeHashtable::longestConsecutive(vector<int>& nums)

{

unordered\_set<int> set;

int max\_length = 0;

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

{

set.insert(nums[i]);

}

while (!set.empty())

{

int length = 1;

int number = \*set.begin();

set.erase(number);

int index = -1;

while (set.find(number + index) != set.end())

{

set.erase(number + index);

length++;

index--;

}

index = 1;

while (set.find(number + index) != set.end())

{

set.erase(number + index);

length++;

index++;

}

max\_length = max(max\_length, length);

}

return max\_length;

}

## 166. Fraction to Recurring Decimal

Medium

Given two integers representing the numerator and denominator of a fraction, return *the fraction in string format*.

If the fractional part is repeating, enclose the repeating part in parentheses.

If multiple answers are possible, return **any of them**.

It is **guaranteed** that the length of the answer string is less than 104 for all the given inputs.

**Example 1:**

**Input:** numerator = 1, denominator = 2

**Output:** "0.5"

**Example 2:**

**Input:** numerator = 2, denominator = 1

**Output:** "2"

**Example 3:**

**Input:** numerator = 2, denominator = 3

**Output:** "0.(6)"

**Example 4:**

**Input:** numerator = 4, denominator = 333

**Output:** "0.(012)"

**Example 5:**

**Input:** numerator = 1, denominator = 5

**Output:** "0.2"

**Constraints:**

* -231 <= numerator, denominator <= 231 - 1
* denominator != 0

### Analysis:

We put the remaining in a hashtable, if we see it is repeated, we know we get to a loop.

/// <summary>

/// Leet code #166. Fraction to Recurring Decimal

/// Given two integers representing the numerator and denominator of a

/// fraction, return the fraction in string format.

/// If the fractional part is repeating, enclose the repeating part in

/// parentheses.

/// For example,

/// Given numerator = 1, denominator = 2, return "0.5".

/// Given numerator = 2, denominator = 1, return "2".

/// Given numerator = 2, denominator = 3, return "0.(6)"

/// Hint:

/// 1.No scary math, just apply elementary math knowledge. Still remember

/// how to perform a long division?

/// 2.Try a long division on 4/9, the repeating part is obvious. Now try

/// 4/333. Do you see a pattern?

/// 3.Be wary of edge cases! List out as many test cases as you can think

/// of and test your code thoroughly.

/// </summary>

string LeetCodeHashtable::fractionToDecimal(int numerator, int denominator)

{

string result;

vector<long long> decimals;

unordered\_map<long long, int> map;

if (denominator == 0) return "overflow";

if (numerator == 0) return "0";

int sign = ((numerator > 0) ^ (denominator > 0)) ? -1 : 1;

if (sign < 0) result = "-";

long long long\_numerator = abs((long long)numerator);

long long long\_denominator = abs((long long)denominator);

long long dividend = long\_numerator / long\_denominator;

long long reminder = long\_numerator % long\_denominator;

result.append(to\_string(dividend));

if (reminder != 0) result.append(".");

int pos = 0;

int repeat\_pos = -1;

while (reminder != 0)

{

if (map.find(reminder) != map.end())

{

repeat\_pos = map[reminder];

break;

}

else

{

map[reminder] = pos;

}

reminder = reminder \* 10;

dividend = reminder / long\_denominator;

decimals.push\_back(dividend);

reminder = reminder % long\_denominator;

pos++;

}

for (size\_t i = 0; i < decimals.size(); i++)

{

if (i == repeat\_pos)

{

result.append("(");

}

result.append(to\_string(decimals[i]));

}

if (repeat\_pos != -1) result.append(")");

return result;

}

## 1915. Number of Wonderful Substrings

<https://leetcode.com/problems/number-of-wonderful-substrings/description/>

/// <summary>

/// Leet code 1915. Number of Wonderful Substrings

///

/// Medium

///

/// A wonderful string is a string where at most one letter appears an odd

/// number of times.

///

/// For example, "ccjjc" and "abab" are wonderful, but "ab" is not.

/// Given a string word that consists of the first ten lowercase English

/// letters ('a' through 'j'), return the number of wonderful non-empty

/// substrings in word. If the same substring appears multiple times in

/// word, then count each occurrence separately.

///

/// A substring is a contiguous sequence of characters in a string.

///

/// Example 1:

/// Input: word = "aba"

/// Output: 4

/// Explanation: The four wonderful substrings are underlined below:

/// - "aba" -> "a"

/// - "aba" -> "b"

/// - "aba" -> "a"

/// - "aba" -> "aba"

///

/// Example 2:

/// Input: word = "aabb"

/// Output: 9

/// Explanation: The nine wonderful substrings are underlined below:

/// - "aabb" -> "a"

/// - "aabb" -> "aa"

/// - "aabb" -> "aab"

/// - "aabb" -> "aabb"

/// - "aabb" -> "a"

/// - "aabb" -> "abb"

/// - "aabb" -> "b"

/// - "aabb" -> "bb"

/// - "aabb" -> "b"

///

/// Example 3:

/// Input: word = "he"

/// Output: 2

/// Explanation: The two wonderful substrings are underlined below:

/// - "he" -> "h"

/// - "he" -> "e"

///

/// Constraints:

/// 1. 1 <= word.length <= 10^5

/// 2. word consists of lowercase English letters from 'a' to 'j'.

/// </summary>

long long LeetCodeHashtable::wonderfulSubstrings(string word)

{

vector<int> str\_count(2048);

str\_count[0] = 1;

int bit\_mask = 0;

long long result = 0;

for (size\_t i = 0; i < word.size(); i++)

{

bit\_mask ^= (1 << (word[i] - 'a'));

result += str\_count[bit\_mask];

for (int j = 0; j < 10; j++)

{

int target\_mask = bit\_mask ^ (1 << j);

result += str\_count[target\_mask];

}

str\_count[bit\_mask] ++;

}

return result;

}

## 2857. Number of Wonderful Substrings

<https://leetcode.com/problems/count-pairs-of-points-with-distance-k/description/>

/// <summary>

/// Leet Code 2857. Count Pairs of Points With Distance k

///

/// Medium

///

/// You are given a 2D integer array coordinates and an integer k, where

/// coordinates[i] = [xi, yi] are the coordinates of the ith point in a

/// 2D plane.

///

/// We define the distance between two points (x1, y1) and (x2, y2) as

/// (x1 XOR x2) + (y1 XOR y2) where XOR is the bitwise XOR operation.

///

/// Return the number of pairs (i, j) such that i < j and the distance

/// between points i and j is equal to k.

///

/// Example 1:

/// Input: coordinates = [[1,2],[4,2],[1,3],[5,2]], k = 5

/// Output: 2

/// Explanation: We can choose the following pairs:

/// - (0,1): Because we have (1 XOR 4) + (2 XOR 2) = 5.

/// - (2,3): Because we have (1 XOR 5) + (3 XOR 2) = 5.

///

/// Example 2:

/// Input: coordinates = [[1,3],[1,3],[1,3],[1,3],[1,3]], k = 0

/// Output: 10

/// Explanation: Any two chosen pairs will have a distance of 0. There

/// are 10 ways to choose two pairs.

/// Constraints:

/// 1. 2 <= coordinates.length <= 50000

/// 2. 0 <= xi, yi <= 10^6

/// 3. 0 <= k <= 100

/// 4. You are given a 0-indexed sorted array of integers nums.

/// </summary>

int LeetCodeHashtable::countPairs(vector<vector<int>>& coordinates, int k)

{

unordered\_map<int, unordered\_map<int, int>> coord\_count;

int result = 0;

for (size\_t i = 0; i < coordinates.size(); i++)

{

int x1 = coordinates[i][0];

int y1 = coordinates[i][1];

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

{

int x2 = i ^ x1;

int y2 = (k - i) ^ y1;

if (coord\_count.count(x2) > 0)

{

if (coord\_count[x2].count(y2) > 0)

{

result += coord\_count[x2][y2];

}

}

}

coord\_count[x1][y1]++;

}

return result;

}