Assignment 3

1. 存在重复元素 I

给你一个整数数组 `nums` 。如果任意一值在数组中出现至少两次，返回 `true` ；如果数组中每个值仅出现一次，返回 `false` 。

示例 1:

- **输入**: nums = [1,2,3,1]
- **输出**: true

示例 2:

- **输入**: nums = [1,2,3,4]
- **输出**: false

示例 3:

- **输入**: nums = [1,1,1,3,3,4,3,2,4,2]
- **输出**: true

提示:

- 1 <= nums.length <= 10^5
- -10^9 <= nums[i] <= 10^9

2. 存在重复元素 II

给你一个整数数组 `nums` 和一个整数 `k` ，判断数组中是否存在两个不同的索引 `i` 和 `j` ，使得 `nums[i] == nums[j]` 并且 `abs(i - j) <= k` 。

示例 1

- 输入: nums = [1,2,3,1], k = 3
- 输出: true

示例 2 :

- 输入: nums = [1,0,1,1], k = 1
- 输出: true

示例 3 :

- 输入: nums = [1,2,3,1,2,3], k = 2
- 输出: false

提示 :

- 1 <= nums.length <= 10^5
- -10^9 <= nums[i] <= 10^9
- 0 <= k <= 10^5

3. 存在重复元素 III

给你一个整数数组 `nums` 和两个整数 `indexDiff` 和 `valueDiff`。

找出是否存在这样的两个下标 `(i, j)`：

- `i != j`
- `abs(i - j) <= indexDiff`
- `abs(nums[i] - nums[j]) <= valueDiff`

如果存在，返回 `true`；否则，返回 `false`。

示例 1 :

- **输入:** nums = [1,2,3,1], indexDiff = 3, valueDiff = 0
- **输出:** true
- **解释:** 可以找出 `(i, j) = (0, 3)`。

满足下述 3 个条件：

i != j ---> i != 3

abs(i - j) <= indexDiff ---> abs(0 - 3) <= 3

abs(nums[i] - nums[j]) <= valueDiff ---> abs(1 - 1) <= 0

示例 2 :

输入: nums = [1,5,9,1,5,9], indexDiff = 2, valueDiff = 3
输出: false
解释: 无法找到满足的下标 `(i, j)`，均无法满足这 3 个条件, 因此返回 `false`。

提示 :

- 2 <= nums.length <= 10^5
- -10^9 <= nums[i] <= 10^9
- 1 <= indexDiff <= nums.length
- 0 <= valueDiff <= 10^9

4. 代码阅读和分析：

请阅读以下 10 个计算 fibonacci 中第 i 个元素的实现函数，并对每个函数回答下列问题：

1)、该实现版本是否正确？如果不正确又什么需要修改的

2)、请说明该实现有什么特点？这种实现方式的主要优点是什么？适合于什么样的场景？

3)、请说明该实现版本中有哪些 c/c++程序设计语法和算法细节是你原来不了解的，请说明这些语法

4)、请按照 Google C++ Coding Style 的要求，为每个函数添加注释

5)、请你在综合这 10 个版本之后，自己重新写一个版本。该版本是你认为最优雅，最好的计算 fibonacci 中第 i 个元素的实现

序号	代码

1	int fibonacci_recursive(int n) {
	  if (n <= 1) return n;
	  return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2);
	}

2	int fibonacci_iterative(int n) {
	  if (n <= 1) return n;
	  int a = 0, b = 1, temp;
	  for (int i = 2; i <= n; ++i) {
	  temp = a + b;
	  a = b;
	  b = temp;
	  }
	  return b;
	}

3	#include <vector>
	int fibonacci_dp(int n) {
	  if (n <= 1) return n;
	  std::vector<int> dp(n + 1, 0);
	  dp[1] = 1;
	  for (int i = 2; i <= n; ++i) {
	  dp[i] = dp[i - 1] + dp[i - 2];
	  }
	  return dp[n];
	}

4	#include <array>
	std::array<std::array<long long, 2>, 2> matrix_multiply(const std::array<std::array<long long, 2>, 2>& a, const std::array<std::array<long long, 2>, 2>& b) {
	  std::array<std::array<long long, 2>, 2> result = {{0}};
	  for (int i = 0; i < 2; ++i)

	for (int j = 0; j < 2; ++j)
	for (int k = 0; k < 2; ++k)
	result[i][j] += a[i][k] * b[k][j];
	return result;
	}
	
	long long fibonacci_matrix(int n) {
	if (n <= 1) return n;
	std::array<std::array<long long, 2>, 2> base = {{1, 1}, {1, 0}};
	std::array<std::array<long long, 2>, 2> result = {{1, 0}, {0, 1}};
	n--;
	while (n > 0) {
	if (n & 1) result = matrix_multiply(result, base);
	base = matrix_multiply(base, base);
	n >>= 1;
	}
	return result[0][0];
	}

5	#include <unordered_map>
	long long fibonacci_memoization(int n, std::unordered_map<int, long long>& memo) {
	if (n <= 1) return n;
	if (memo.find(n) != memo.end()) return memo[n];
	memo[n] = fibonacci_memoization(n - 1, memo) + fibonacci_memoization(n - 2, memo);
	return memo[n];
	}

6	#include <optional>
	std::optional<long long> fibonacci_safe(int n) {
	if (n < 0) return std::nullopt;
	if (n <= 1) return n;
	long long a = 0, b = 1, temp;
	for (int i = 2; i <= n; ++i) {
	temp = a + b;
	a = b;
	b = temp;
	}
	return b;
	}

7	constexpr long long fibonacci_constexpr(int n) {
	if (n <= 1) return n;
	long long a = 0, b = 1;
	for (int i = 2; i <= n; ++i) {
	long long temp = a + b;

	a = b;
	b = temp;
	}
	return b;
	}

8	#include <future>
	long long fibonacci_parallel(int n) {
	if (n <= 1) return n;
	auto future = std::async(std::launch::async, fibonacci_parallel, n - 2);
	long long result = fibonacci_parallel(n - 1);
	return result + future.get();
	}

9	#include <boost/multiprecision/cpp_int.hpp>
	boost::multiprecision::cpp_int fibonacci_bigint(int n) {
	if (n <= 1) return n;
	boost::multiprecision::cpp_int a = 0, b = 1, temp;
	for (int i = 2; i <= n; ++i) {
	temp = a + b;
	a = b;
	b = temp;
	}
	return b;
	}

10	template<int N, typename = std::enable_if_t<(N >= 0)>>
	struct Fibonacci {
	static constexpr long long value = Fibonacci<N-1>::value + Fibonacci<N-2>::value;
	};
	
	template<>
	struct Fibonacci<0> {
	static constexpr long long value = 0;
	};
	
	template<>
	struct Fibonacci<1> {
	static constexpr long long value = 1;
	};