算法分析和複雜性理論

于皓丞，2101212850, 信息工程學院

2022 年 3 月 1 日

1 作業目標與章節摘要

1. LeetCode 熟悉
2. LeetCode 1. Two Sum, 兩數之和
3. LeetCode 69. Sqrt(x), x 的平方根
4. LeetCode 70. Climbing Stairs, 爬樓梯

2 作業內容概述

作業可以從 GitHub 下的 kancheng/kan-cs-report-in-2022 專案找到，作業程式碼與文件目錄為 kan-cs-report-in-2022/AATCC/lab-report/w1。實際執行的環境與實驗設備為 Google 的 Colab 、MacBook Pro (Retina, 15-inch, Mid 2014) 、Acer Aspire R7 與 HP Victus (Nvidia GeForce RTX 3060)。

本作業 GitHub 專案為 kancheng/kan-cs-report-in-2022 下的 AATCC‘ 的目錄。程式碼可以從 code 目錄下可以找到 *.pynb，內容包含上次課堂練習、LeetCode 範例思路整理與作業，最後包含其他語言範例。

3 LeetCode 熟悉

1. LeetCode : https://leetcode.com/ 2. LeetCode CN : https://leetcode-cn.com/
LeetCode 的平台部分，CN 的平台有針對簡體中文使用者進行處理，包含中英文切換等功能。

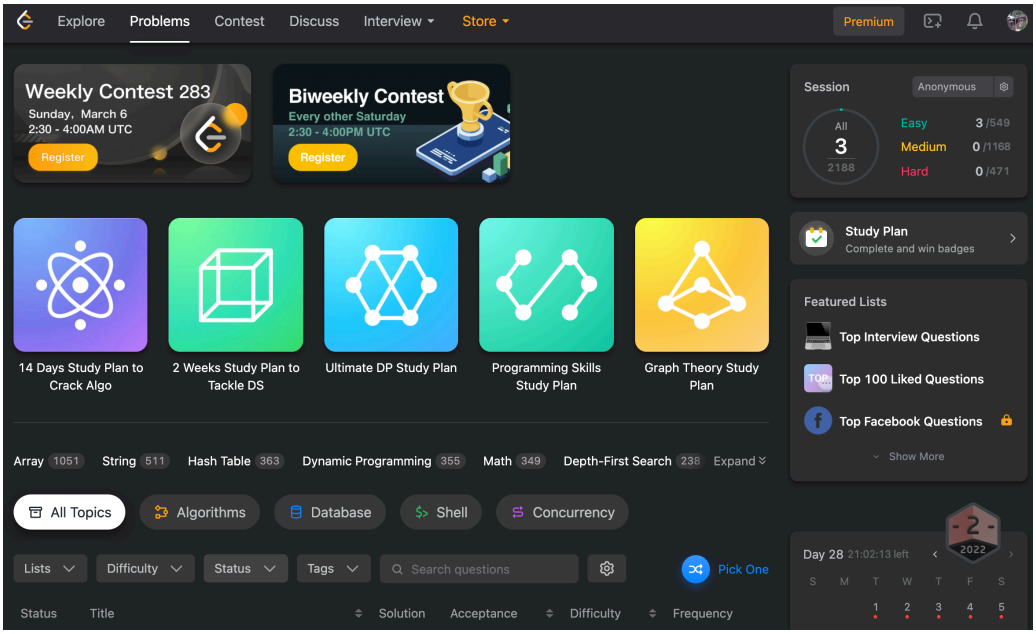

Fig. 1. LeetCode

4 LeetCode 1. Two Sum, 兩數之和

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

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

You can return the answer in any order.

給定一個整數數組 nums 和一個整數目標值 target，請你在該數組中找出和為目標值 target 的那兩個整數，並返回它們的數組下標。

你可以假設每種輸入只會對應一個答案。但是，數組中同一個元素在答案裡不能重複出現。

你可以按任意順序返回答案。

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

LeetCode 1. **思路總結**

1. **用** For **將每個元素讀過一遍，然後將其逐一取出來一個個判斷，若目標為** 9**，找到元素** 2 **，就會找** 7**，若找到元素** 7 **，就會找** 2**。效率上沒有很理想。**Java、C、C++ **可以用陣列等等的方式。**

2. **運用** Python **的字典可以直接去找，該原理與** Hash Map **類似。用** For **去找，剩下用** IF **來判斷該值有沒有在字典裡面。基本上不論用何種程式語言，只要有用上** Hash Map **類似的原理就會有比較理想的結果。**

LeetCode 1. Python 1

```
class Solution(object):
    def twoSum(self, nums, target):
        required = {}
        for i in range(len(nums)):
            if target - nums[i] in required:
                return [required[target - nums[i]],i]
            else:
                required[nums[i]]=i
input_list = [ 2, 7, 11, 15]
target = 9
ob1 = Solution()
print(ob1.twoSum(input_list, target))
```

Runtime: 139 ms, faster than 38.38% of Python3 online submissions for Two Sum. Memory Usage: 15.2 MB, less than 41.73% of Python3 online submissions for Two Sum.

LeetCode 1. Python 2

```
class Solution(object):
    def twoSum(self, nums, target):
```

```
    for i in range(len(nums)):
        tmp = nums[i]
        remain = nums[i+1:]
        if target - tmp in remain:
            return[i, remain.index(target - tmp)+ i + 1]
input_list = [ 2, 7, 11, 15]
target = 9
ob1 = Solution()
print(ob1.twoSum(input_list , target))
```

Runtime: 707 ms, faster than 35.03% of Python3 online submissions for Two Sum. Memory Usage: 14.9 MB, less than 73.00% of Python3 online submissions for Two Sum.

LeetCode 1. Python 3

```
class Solution(object):
    def twoSum(self, nums, target):
        dict = {}
        for i in range(len(nums)):
            if target - nums[i] not in dict:
                dict[nums[i]] = i
            else:
                return [dict[target - nums[i]], i]
input_list = [ 2, 7, 11, 15]
target = 9
ob1 = Solution()
print(ob1.twoSum(input_list , target))
```

Runtime: 64 ms, faster than 86.00% of Python3 online submissions for Two Sum. Memory Usage: 15.2 MB, less than 57.63% of Python3 online submissions for Two Sum.

LeetCode 1. Java 1

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        int l = nums.length;
        int[] ans = new int[2];
        int i,j;
        for( i = 0; i < l - 1; i++) {
            for( j = i + 1 ; j < l; j++) {
                if( nums[i] + nums[j] == target)
                {
                    ans[0] = i;
                    ans[1] = j;
                }
            }
        }
        return ans;
    }
}
```

Runtime: 68 ms, faster than 29.45% of Java online submissions for Two Sum. Memory Usage: 45.3 MB, less than 26.94% of Java online submissions for Two Sum.

LeetCode 1. Java 2

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> numMap = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (numMap.containsKey(complement)) {
                return new int[] { numMap.get(complement), i };
            } else {
                numMap.put(nums[i], i);
            }
        }
        return new int[] {};
    }
}
```

Runtime: 2 ms, faster than 90.72% of Java online submissions for Two Sum. Memory Usage: 46.1 MB, less than 10.02% of Java online submissions for Two Sum.

LeetCode 1. C++

```cpp
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int,int> record;
        for(int i = 0 ; i < nums.size() ; i ++){
            int complement = target - nums[i];
            if(record.find(complement) != record.end()){
                int res[] = {i, record[complement]};
                return vector<int>(res, res + 2);
            }
            record[nums[i]] = i;
        }
        return {};
    }
};
```

Runtime: 11 ms, faster than 88.73% of C++ online submissions for Two Sum. Memory Usage: 10.8 MB, less than 51.52% of C++ online submissions for Two Sum.

LeetCode 1. C

```c
int* twoSum(int* nums, int numsSize, int target, int* returnSize) {
    int *ans=(int *)malloc(2 * sizeof(int));
    int i,j;
    bool flag=false;
    for(i=0;i<numsSize-1;i++) {
        for(j=i+1;j<numsSize;j++) {
            if(nums[i]+nums[j] == target) {
```

```
                    ans[0]=i;
                    ans[1]=j;
                    flag=true;
                }
            }
        }
        if(flag) {
            *returnSize = 2;
        } else {
            *returnSize = 0;
        }
        return ans;
}
```

Runtime: 116 ms, faster than 65.84% of C online submissions for Two Sum. Memory Usage: 6.5 MB, less than 46.99% of C online submissions for Two Sum.

LeetCode 1. Go

```
func twoSum(nums []int, target int) []int {
    m := make(map[int]int)
    for i := 0; i < len(nums); i++ {
        another := target - nums[i]
        if _, ok := m[another]; ok {
            return []int{m[another], i}
        }
        m[nums[i]] = i
    }
    return nil
}
```

Runtime: 12 ms, faster than 54.81% of Go online submissions for Two Sum. Memory Usage: 4.4 MB, less than 38.96% of Go online submissions for Two Sum.

5 LeetCode 69. Sqrt(x), x 的平方根

Given a non-negative integer x, compute and return the square root of x.

Since the return type is an integer, the decimal digits are truncated, and only the integer part of the result is returned.

Note: You are not allowed to use any built-in exponent function or operator, such as pow(x, 0.5) or x ** 0.5.

給你一個非負整數 x ，計算並返回 x 的算術平方根。

由於返回類型是整數，結果只保留整數部分，小數部分將被捨去。

注意：不允許使用任何內置指數函數和算符，例如 pow(x, 0.5) **或者** x ** 0.5 **。**

Example 1:

```
Input: x = 4
Output: 2
```

Example 2:

```
Input: x = 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since the decimal part is
    truncated, 2 is returned.
```

LeetCode 69. 思路總結

1. 二分法, 找到最後一個滿足 n2 <= x 的整□ n

2. 使用牛頓迭代法，也就是微積分的方式來進行處理

LeetCode 69. Python

```
class Solution:
    def mySqrt(self, x):
        """
        :type x: int
        :rtype: int
        """
        if x < 2:
            return x
        left, right = 1, x // 2
        while left <= right:
            mid = left + (right - left) // 2
            if mid > x / mid:
                right = mid - 1
            else:
                left = mid + 1
        return left - 1
x1 = 4
x2 = 9
ob1 = Solution()
print(ob1.mySqrt(x1))
print(ob1.mySqrt(x2))
```

Runtime: 60 ms, faster than 47.79% of Python3 online submissions for Sqrt(x). Memory Usage: 13.9 MB, less than 81.69% of Python3 online submissions for Sqrt(x).

LeetCode 69. Go 1

```
func mySqrt(x int) int {
    l, r := 0, x
    for l < r {
        mid := (l + r + 1) / 2
        if mid*mid > x {
            r = mid - 1
        } else {
            l = mid
        }
    }
    return l
}
```

Runtime: 0 ms, faster than 100.00% of Go online submissions for Sqrt(x). Memory Usage: 2 MB, less than 88.97% of Go online submissions for Sqrt(x).

LeetCode 69. Go 2

```
func mySqrt(x int) int {
    r := x
    for r*r > x {
        r = (r + x/r) / 2
    }
    return r
}
```

Runtime: 4 ms, faster than 55.90% of Go online submissions for Sqrt(x). Memory Usage: 2.1 MB, less than 88.97% of Go online submissions for Sqrt(x).

6 LeetCode 70. Climbing Stairs, 爬樓梯

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

假設你正在爬樓梯。需要 n 階你才能到達樓頂。

每次你可以爬 1 或 2 個台階。你有多少種不同的方法可以爬到樓頂呢？

Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

Example 2:

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

LeetCode 70. **思路總結**

1. **動態規劃，遞迴公式** $f(n-1)+f(n-2)$**，其結果就是費氏數列。**

2. **使用牛頓迭代法，也就是微積分的方式來進行處理。**

LeetCode 70. Python

```
class Solution:
    def climbStairs(self, n):
        prev, current = 0, 1
        for i in range(n):
            prev, current = current, prev + current
        return current
x1 = 2
```

```
x2 = 3
ob1 = Solution()
print(ob1.climbStairs(x1))
print(ob1.climbStairs(x2))
```

Runtime: 46 ms, faster than 40.45% of Python3 online submissions for Climbing Stairs. Memory Usage: 13.8 MB, less than 82.27% of Python3 online submissions for Climbing Stairs.

LeetCode 70. JS

```
var climbStairs = function(n) {
    let temp = new Array(n+1);
    temp[1] = 1;
    temp[2] = 2;
    for (let i = 3; i < temp.length; i++) {
        temp[i] = temp[i-1] + temp[i-2];
    }
    return temp[n];
}
```

Runtime: 131 ms, faster than 5.35% of JavaScript online submissions for Climbing Stairs. Memory Usage: 42 MB, less than 22.19% of JavaScript online submissions for Climbing Stairs.

LeetCode 70. Go

```
func climbStairs(n int) int {
    dp := make([]int, n+1)
    dp[0], dp[1] = 1, 1
    for i := 2; i <= n; i++ {
        dp[i] = dp[i-1] + dp[i-2]
    }
    return dp[n]
}
```

Runtime: 0 ms, faster than 100.00% of Go online submissions for Climbing Stairs. Memory Usage: 2 MB, less than 66.83% of Go online submissions for Climbing Stairs.