Алгоритмын шинжилгээ ба зохиомж
Лабораторийн ажил
(F.CS301)

B210900007 Н.Тэмүүлэн

2024/12/2

Агуулга

1 1-р хичээл **2**

2 3-р хичээл **4**

3 4-р хичээл **6**

4 5-р хичээл **8**

5 6-р хичээл **9**

6 7-р хичээл **10**

7 11-р хичээл **11**

1 1-р хичээл

- Two Sum

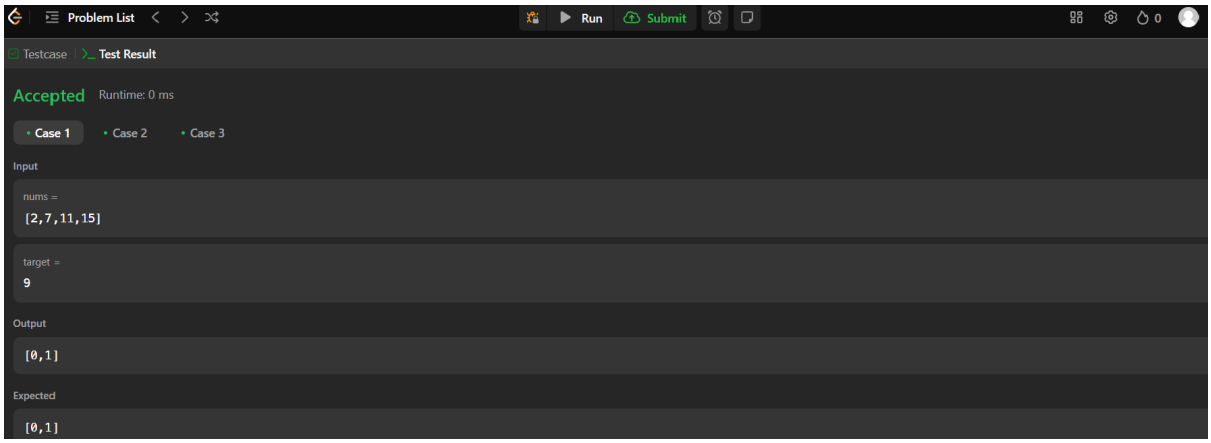
```
class Solution:
def twoSum(self, nums, target):
    num_dict = {}

    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_dict:
            return [num_dict[complement], i]

        num_dict[num] = i
    raise ValueError("No solution found")
```

Тайлбар

Энэ бодлого дээр `dictionary` ашиглан массивын элементүүдийн утгууд болон тэдгээрийн индексийг хадгална. Ингэснээр зөвхөн нэг удаа массивыг гүйлгэж, шаардлагатай индексуудыг хурдан олох боломжтой.



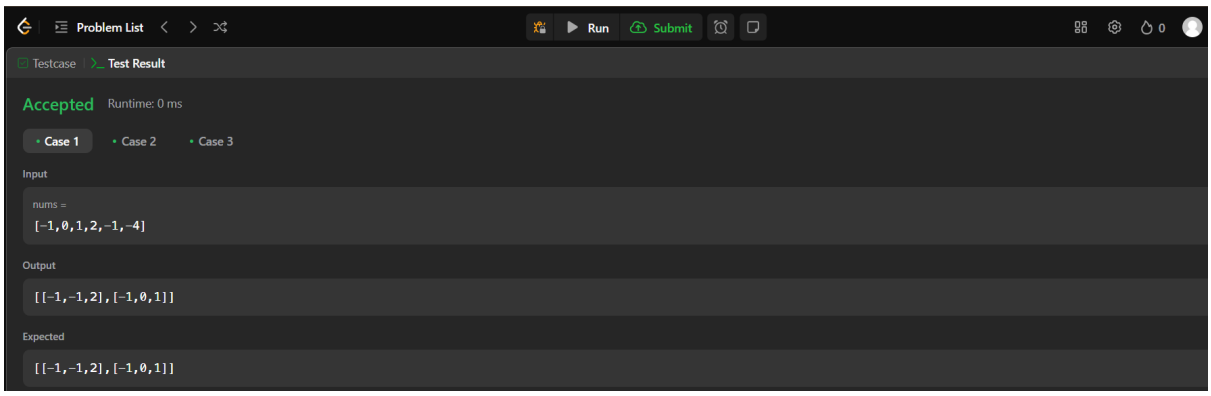
Зураг 1: Test result

- 3 Sum

```
class Solution:
    def threeSum(self, nums):
        nums.sort()
        result = []
        for i in range(len(nums) - 2):
            if i > 0 and nums[i] == nums[i - 1]: continue
            left, right = i + 1, len(nums) - 1
            while left < right:
                s = nums[i] + nums[left] + nums[right]
                if s == 0:
                    result.append([nums[i], nums[left], nums[right]])
                    while left < right and nums[left] == nums[left +
                        1]: left += 1
                    while left < right and nums[right] == nums[right -
                        1]: right -= 1
                    left += 1
                    right -= 1
                elif s < 0:
                    left += 1
                else:
                    right -= 1
        return result
```

Тайлбар

`Sorting` болон `two-pointer` техникийг ашигласан. Эхлээд массивыг эрэмбэлж, нэг элементийг байршуулах замаар хоёр нэмэлт индексийг хайж, дараа нь шаардлагатай нөхцлийг хангаж буй гурван элементийг олно.



Зураг 2: Test result

2 3-р хичээл

- Majority Element

```java
import java.util.*;

class Solution {
    public int majorityElement(int[] nums) {
        int count = 0;
        int candidate = 0;

        for (int num : nums) {
            if (count == 0) {
                candidate = num;
            }
            count += (num == candidate) ? 1 : -1;
        }

        return candidate;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] nums1 = {3, 2, 3};
        int[] nums2 = {2, 2, 1, 1, 1, 2, 2};

        System.out.println("Majority Element in nums1: " + solution
            .majorityElement(nums1));
        System.out.println("Majority Element in nums2: " + solution
            .majorityElement(nums2));
    }
}
```

Тайлбар

Өгөгдсөн массив доторх ихэнх элемент буюу $n/2$-оос олон тоогоор давтагдсан элементийг олох зорилготой. Үүнийг Boyer-Moore Voting Algorithm ашиглан хэрэгжүүлсэн. Хэрэв `count == 0` байвал дараагийн элементийг `candidate` болгон сонгоно, дараагийн элемент нь `candidate`-тай тэнцүү бол `count`-г нэмнэ, үгүй бол хасна. Давталт дуусахад `candidate` хувьсагчид ихэнх элемент хадгалагдсан байна.

- Longest Nice Substring

```
class Solution:
    def longestNiceSubstring(self, s: str) -> str:
    if len(s) < 2:a
        return ""
    char_set = set(s)
    for i, char in enumerate(s):
        if char.swapcase() not in char_set:
            left = self.longestNiceSubstring(s[:i])
            right = self.longestNiceSubstring(s[i + 1:])
            return left if len(left) >= len(right) else right
    return s
```

Тайлбар

Энэ код нь өгөгдсөн тэмдэгт мөрөөс хамгийн урт nice дэд мөрийг олох зорилготой. Nice дэд мөр гэдэг нь тухайн мөрөнд байгаа бүх тэмдэгтүүдийн хувьд том болон жижиг үсгийн аль аль нь оршиж байгааг хэлнэ. Код нь рекурсив аргаар тэмдэгт мөрийг хуваан, тухайн тэмдэгтүүдийн нөхцөл эвдсэн хэсгийг дахин шалгаж хамгийн урт nice дэд мөрийг буцаадаг.

- Sort an Array

```
class Solution:
    def sortArray(self, nums: list[int]) -> list[int]:
        if len(nums) <= 1:
            return nums
    def merge(left, right):
        sorted_array = []
        i = j = 0
        while i < len(left) and j < len(right):
            if left[i] <= right[j]:
                sorted_array.append(left[i])
                i += 1
            else:
                sorted_array.append(right[j])
                j += 1
        sorted_array.extend(left[i:])
        sorted_array.extend(right[j:])

        return sorted_array
    mid = len(nums) // 2
    left = self.sortArray(nums[:mid])
    right = self.sortArray(nums[mid:])
    return merge(left, right)
```

Тайлбар

Merge Sort алгоритмыг ашиглан тоонуудын массивыг өсөх дарааллаар эрэмбэлэхэд зориулагдсан. Алгоритм нь массивыг хувааж, дахин ялгаж, эцэст нь ялгагдсан хэсгүүдийг нэгтгэн эрэмбэлсэн массив үүсгэдэг. Merge Sort нь $O(n \log n)$ цагийн complexity-тэй бөгөөд массивын бүх элементийг харьцуулж, зөв байрлуулахын тулд нэмэлт санах ой шаарддаг.

- Convert Sorted Array to Binary Search Tree

```
class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> Optional[
        TreeNode]:
        if not nums:
            return None
        mid = len(nums) // 2
        return TreeNode(nums[mid], self.sortedArrayToBST(nums[:mid
            ]), self.sortedArrayToBST(nums[mid + 1:]))
```

Тайлбар

Эрэмбэлэгдсэн массивыг балансласгдсан хоёр дахь мод (BST) болгон хувиргахад зориулагдсан. Массивын дунд байрлах элементийг үндсэн зангилаа болгож, зүүн болон баруун талын хэсгийг дахин ижил аргаар мод болгодог. Ингэснээр мод нь хамгийн бага өндөртэй, тэгш хуваасан бүтэцтэй болно.

3 4-р хичээл

- Climbing Stairs

```
class Solution:
    def climbStairs(self, n: int) -> int:
        if n == 1:
            return 1
        elif n == 2:
            return 2
        first = 1
        second = 2

        for i in range(3, n + 1):
            current = first + second
            first = second
```

```
            second = current

        return second
```

Тайлбар

n шаттай шатаар өгөгдсөн тоогоор хамгийн их 1 болон 2 алхмаар гарч болох олон янзын аргыг тоолох зориулалттай. Хэрэв $n = 1$ бол нэг арга, $n = 2$ бол хоёр арга байдаг. Дараа нь 3-р шатнаас эхлэн, бүх шатанд хүрэх аргуудыг өмнөх хоёр шатны аргыг нэмж тооцоолсон. Ингэснээр Фибоначчийн дараалалтай адил тооцоолол хийгддэг. Энэ нь $O(n)$ цагийн complexity-тэй бөгөөд $O(1)$ санах ой шаарддаг.

- Min Cost Climbing Stairs

```
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:
        if len(cost) == 2:
            return min(cost[0], cost[1])
        first = cost[0]
        second = cost[1]
        for i in range(2, len(cost)):
            current = cost[i] + min(first, second)
            first = second
            second = current
        return min(first, second)
```

Тайлбар

Динамик программчлалын аргаар шатны хамгийн бага зардлыг тооцоолно. Өмнөх хоёр шатны зардлыг хадгалж, дараагийн шатны зардлыг хамгийн бага зардлыг сонгоно. Эцэст нь хамгийн бага зардлыг буцаадаг.

- Coin Change

```
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        dp = [float('inf')] * (amount + 1)
        dp[0] = 0
        for coin in coins:
            for i in range(coin, amount + 1):
                dp[i] = min(dp[i], dp[i - coin] + 1)
        return dp[amount] if dp[amount] != float('inf') else -1
```

Динамик програмчлалын аргаар хамгийн бага зардалтай мөнгөний хэмжээ олохыг зорьдог. Массивт мөнгөний хэмжээ тус бүрт шаардлагатай хамгийн бага зардлыг хадгалж, аль болох бага тооны зардалд хүрэхийн тулд бүх боломжит зээлүүдийг тооцоолдог. Хэрэв хүссэн хэмжээг гаргах боломжгүй бол -1 буцаана.

- House Robber

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        if not nums:
            return 0
        if len(nums) == 1:
            return nums[0]
        prev2 = nums[0]
        prev1 = max(nums[0], nums[1])
        for i in range(2, len(nums)):
            current = max(prev1, prev2 + nums[i])
            prev2 = prev1
            prev1 = current
        return prev1
```

Хөрш хоёр байшингийн хамгийн их хулгайлах мөнгийг тооцоолно. Хөрш байшингуудыг хулгайлах боломжгүй тул хамгийн их мөнгийг хадгалах хоёр хувьсагчийг ашиглан байшин бүрийн хамгийн их мөнгийг олж, үргэлжлүүлэн шинэчлээд эцэст нь хамгийн их мөнгийг буцаана.

4 5-р хичээл

- Longest Common Subsequence

```
class Solution:
    def longestCommonSubsequence(self, text1: str, text2: str) ->
        int:
        dp = [[0] * (len(text2) + 1) for _ in range(len(text1) + 1)
            ]
        for i in range(1, len(text1) + 1):
            for j in range(1, len(text2) + 1):
                if text1[i - 1] == text2[j - 1]:
                    dp[i][j] = dp[i - 1][j - 1] + 1
                else:
                    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
        return dp[len(text1)][len(text2)]
```

Хоёр текстийн хамгийн урт нийтлэг дэд утгыг олохын тулд динамик программчлалыг ашиглах ба 2 хэмжээст массив ашиглан текстийн бүх боломжит хослолуудыг үнэлж, хамгийн урт дэд утгыг олно. Эцэст нь массивын хамгийн сүүлчийн элемент нь хамгийн урт дэд утгын уртыг буцаана.

5 6-р хичээл

- Unique Binary Search Trees

```
class Solution:
    def numTrees(self, n: int) -> int:
        dp = [0] * (n + 1)
        dp[0], dp[1] = 1, 1
        for i in range(2, n + 1):
            for j in range(1, i + 1):
                dp[i] += dp[j - 1] * dp[i - j]
        return dp[n]
```

Динамик программчлалын аргыг ашиглан n тооны Unique Binary хайлтын модны (BST) тоог тооцоолох бөгөөд энэ нь Каталан тоо ашиглан тооцогддог. dp массивыг үүсгэж, $dp[i]$ нь i тооны BST-ийн тоог хадгалдаг. i-г үндсийн хувьд сонгосон тохиолдолд зүүн болон баруун модны тоог dp массив ашиглан тооцоолон нийт тоог гаргана. $dp[n]$ нь n тооны модны тоог буцаана.

- Эрэмбэлэгдсэн n ширхэг хайлтын түлхүүр агуулсан k жагсаалт болон түлхүүр бүрийн хайгдсан тоог агуулсан f жагсаалт өгөгджээ. Тэгвэл нийт түлхүүрийн хувьд хайлтын хамгийн бага өртгийг ол. Жишээлбэл, $n = 2$, $k = \{5, 6\}$, $f = \{17, 25\}$ бол хамгийн бага өртөг нь 59. Учир нь оройн утга нь 5 (5–6) бол $C = 17 \cdot 1 + 25 \cdot 2 = 67$ болно. Харин 6 бол (6–5) $C = 25 \cdot 1 + 17 \cdot 2 = 59$ болно.

```
public class Main {
    public static void main(String[] args) {
        int[] k = {5, 6};
        int[] f = {17, 25};
        int minCost = findMinCost(k, f);

        System.out.println("MinCost: " + minCost);
    }
    public static int findMinCost(int[] k, int[] f) {
        int n = k.length;
        int minCost = Integer.MAX_VALUE;
        for (int i = 0; i < n; i++) {
            int cost = 0;
            for (int j = 0; j < n; j++) {
```

```
                if (i == j) {
                    cost += f[j] * 1;
                } else {
                    cost += f[j] * 2;
                }
            }
            minCost = Math.min(minCost, cost);
        }
        return minCost;
    }
}
```

```
Хайлтын хамгийн бага өртөг: 59
```

- Best Time to Buy and Sell Stock II

```
class Solution:
    def maxProfit(self, prices: list[int]) -> int:
        total_profit = 0
        for i in range(1, len(prices)):
            if prices[i] > prices[i - 1]:
                total_profit += prices[i] - prices[i - 1]
        return total_profit
```

prices жагсаалтад өгөгдсөн хувьцааны үнэд тулгуурлан хамгийн их ашиг тооцоолдог. Хэрэв өнөөдрийн үнэ өчигдрийнхөөс өндөр бол ашиг нэмэгдүүлнэ.Үүний тулд бүх үнийн өдрийг дараалсан байдлаар харьцуулж, ашигтай үед зөрүүг нийт ашигт нэмнэ. Эцэст нь, бүх боломжит өсөлтөөс хуримтлагдсан нийт ашгийг буцаана.

6 7-р хичээл

- Task Scheduler

```
class Solution:
    def leastInterval(self, tasks, n):
        task_counts = Counter(tasks)
        max_freq = max(task_counts.values())
        max_freq_count = sum(1 for task, freq in task_counts.items
            () if freq == max_freq)

        part_count = max_freq - 1
        part_length = n - (max_freq_count - 1)
        empty_slots = part_count * part_length
```

```
        available_tasks = len(tasks) - max_freq * max_freq_count
        idles = max(0, empty_slots - available_tasks)

        return len(tasks) + idles
```

CPU дээрх үүргүүдийг (жишээ нь: A, B гэх мэт) гүйцэтгэх хамгийн бага хугацааг тооцоолдог. Үүргүүдийг гүйцэтгэхэд хооронд нь тодорхой хугацааны (n интервал) завсарлага шаардагдана. Код нь хамгийн их давтамжтай үүргийг олох, завсарлага болон чөлөөт хугацааг тооцоолох замаар нийт хугацааг гаргаж өгнө.

CPU Task Scheduling Result

```
Input: tasks = ["A", "A", "A", "B", "B", "B"], n = 2
Output: 8

Explanation:
A possible sequence is:
A -> B -> idle -> A -> B -> idle -> A -> B
```

7 11-р хичээл

- Island Perimeter

```
class Solution:
    def islandPerimeter(self, grid: List[List[int]]) -> int:
        perimeter = 0

        rows = len(grid)
        cols = len(grid[0])
        for i in range(rows):
            for j in range(cols):
                if grid[i][j] == 1:
                    perimeter += 4
                    if i > 0 and grid[i-1][j] == 1:
                        perimeter -= 1
                    if i < rows - 1 and grid[i+1][j] == 1:
                        perimeter -= 1
                    if j > 0 and grid[i][j-1] == 1:
                        perimeter -= 1
                    if j < cols - 1 and grid[i][j+1] == 1:
                        perimeter -= 1
```

```
        return perimeter
```

Доорх үр дүн нь islandPerimeter функцийн хамгийн бага периметрийг тооцоолсны дараа гарсан утгуудыг харуулж байна.

```
Grid1: [[0,1,0,0],[1,1,1,0],[0,1,0,0],[1,1,0,0]]
Grid2: [[1]]
Grid3: [[1,0]]
Grid1-ийн хувьд периметр: 16
Grid2-ийн хувьд периметр: 4
Grid3-ийн хувьд периметр: 4
```

- Valid Parenthesis String

```
class Solution:
    def checkValidString(self, s: str) -> bool:
        min_open = 0
        max_open = 0

        for char in s:
            if char == '(':
                min_open += 1
                max_open += 1
            elif char == ')':
                min_open -= 1
                max_open -= 1
            else:
                min_open -= 1
                max_open += 1

            if min_open < 0:
                min_open = 0
            if max_open < 0:
                return False
        return min_open == 0
```

- Open the Lock

```
class Solution:
    def openLock(self, deadends, target: str) -> int:
        deadends = set(deadends)

        if "0000" in deadends:
            return -1
        queue = deque([("0000", 0)])
        visited = set(["0000"])
        while queue:
```

```
            current_state, steps = queue.popleft()
            if current_state == target:
                return steps

            for i in range(4):
                for diff in [-1, 1]:
                    next_state = list(current_state)
                    next_state[i] = str((int(next_state[i]) + diff)
                        % 10)
                    next_state = ''.join(next_state)
                    if next_state not in visited and next_state not
                        in deadends:
                        visited.add(next_state)
                        queue.append((next_state, steps + 1))
        return -1
```

0000 гэдэг анхны кодтой эхэлнэ. deadends нэртэй жагсаалт дахь кодуудыг хулгайлахгүйгээр орхиж, ямар ч deadend эсвэл хүрч чадахгүй кодыг орохгүй болгоно. queue гэдэг жагсаалтад одоогийн байдал болон алхмын тоо хамт байрлуулна. Одоогийн байдал бүрээс дараагийн байдал уруу орж, бүх боломжит эргэлтүүдийг туршиж үзнэ. Зорилтот утга буюу target-ийг олоход хүрэхээр BFS ашиглан бүх боломжит эргэлтүүдийг хайна.

- Course Schedule II

```
class Solution:
    def findOrder(self, numCourses: int, prerequisites: List[List[
        int]]) -> List[int]:
        adj_list = defaultdict(list)
        in_degree = [0] * numCourses
        for course, prereq in prerequisites:
            adj_list[prereq].append(course)
            in_degree[course] += 1
        queue = deque()
        for i in range(numCourses):
            if in_degree[i] == 0:
                queue.append(i)
        order = []
        while queue:
            course = queue.popleft()
            order.append(course)
            for next_course in adj_list[course]:
                in_degree[next_course] -= 1
                if in_degree[next_course] == 0:
                    queue.append(next_course)
        return order if len(order) == numCourses else []
```

Энд курсийн урсгал буюу хэдэн курс өөрсдийн урьдчилсан курсүүдтэй холбоотой байгааг олж, ямар дараалалтайгаар тэдгээр курсийг судлах боломжтойг тодорхойлно. Графын өгөгдлийг боловсруулж, циклийн байх эсэхийг шалгаж, зөв дараалалд курсүүдийг жагсаах үүрэгтэй. Хэрвээ боловсруулсан курсүүдийн тоо бүх курсийн тоотой тэнцүү бол зөв дараалал гарсан гэж үзнэ. Хэрэв тийм биш бол цикл үүссэн гэж үзээд хоосон жагсаалт буцаана.

```
[0, 1, 2, 3]
```