

Table of Contents

- [1. Two Sum](#)
- [15. 3Sum](#)
- [19. Remove Nth Node From End of List](#)
- [2. Add Two Numbers](#)
- [206. Reverse Linked List](#)
- [Invert Binary Tree](#)
- [4. Median of Two Sorted Arrays](#)
- [5. Longest Palindromic Substring](#)
- [704. Binary Search](#)
- [94. Binary Tree Inorder Traversal](#)
- [98. Validate Binary Search Tree](#)
- [99. Recover Binary Search Tree](#)
- [21. Merge Two Sorted Lists](#)
- [70. Climbing Stairs](#)
- [9. Palindrome Number](#)
- [18. 4Sum](#)
- [23. Merge k Sorted Lists](#)
- [2529. Maximum Count of Positive Integer and Negative Integer](#)
- [2540. Minimum Common Value](#)
- [35. Search Insert Position](#)
- [876. Middle of the Linked List](#)
- [14. Longest Common Prefix](#)
- [20. Valid Parentheses](#)
- [26. Remove Duplicates from Sorted Array](#)
- [39. Combination Sum](#)
- [46. Permutations](#)
- [47. Permutations II](#)
- [121. Best Time to Buy and Sell Stock](#)
- [136. Single Number](#)
- [1480. Running Sum of 1d Array](#)
- [169. Majority Element](#)
- [191. Number of 1 Bits](#)
- [2094. Finding 3-Digit Even Numbers](#)
- [27. Remove Element](#)
- [58. Length of Last Word](#)
- [628. Maximum Product of Three Numbers](#)
- [88. Merge Sorted Array](#)
- [1768. Merge Strings Alternately](#)
- [28. Find the Index of the First Occurrence in a String](#)
- [189. Rotate Array](#)
- [30. Substring with Concatenation of All Words](#)
- [494. Target Sum](#)
-

- [17. Letter Combinations of a Phone Number](#)
- [42. Trapping Rain Water](#)
- [48. Rotate Image](#)
- [Hash table](#)
- [Interview method](#)
- [Algorithm](#)
- [Study process](#)
- [String](#)

1. Two Sum

layout: post title: 1. Two Sum.md category: leetcode date: 2024-12-18 16:18:45 +0900 description: <https://leetcode.com/problems/two-sum/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

1. Two Sum

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.

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]`

Solution 1.

hash map인 **unordered_map** 이용해서 푸는 방법.

인덱스를 알고 접근하기 때문에

시간 복잡도 $O(n)$

공간 복잡도 $O(n)$

```
#include <vector>
#include <unordered_map>
#include <algorithm>
using namespace std;

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

Solution2. 2 Pointer 기법

정렬 후에 양 쪽 끝에서 sum과 target을 비교하면서

left right를 조정하며 푸는 방법

시간 복잡도

- 정렬에 $O(n \log n)$
- 합 계산에 $O(n)$

공간 복잡도 $O(1)$

특징

1. 포인터 사용

투 포인터 기법에서는 배열의 시작과 끝을 가리키는 두 개의 포인터(변수)만 사용합니다.

예를 들어 left와 right라는 두 변수를 사용하여 배열의 처음과 끝에서부터 이동합니다.

이러한 포인터들 자체는 단지 변수일 뿐이므로, 고정된 크기로 존재합니다.

따라서 추가적으로 사용하는 메모리는 포인터 두 개의 공간밖에 요구되지 않아

상수 시간, 즉 $O(1)$ 의 공간을 차지하게 됩니다.

2. 추가 메모리 사용 없음

투 포인터 기법은 배열 자체를 수정하지 않습니다.

단순히 배열 내의 두 위치를 가리키는 두 개의 포인터만 이동하면서 조건을 확인합니다.

배열 내에서 요소를 비교하거나 합산하는 과정이므로,

별도의 추가적인 데이터 구조나 저장공간이 필요하지 않습니다.

3. 정렬된 배열에서 직접 탐색

정렬된 배열 내에서 포인터를 움직이면서

두 숫자의 합을 목표 값과 비교하여 탐색하는 방식입니다.

정렬 자체는 별도의 단계이기 때문에,

순수한 투 포인터 기법의 공간 복잡도에는 영향을 미치지 않습니다.

```
#include <vector>
#include <algorithm>
using namespace std;

class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        vector<pair<int, int>> sortedNums;
        for (int i = 0; i < nums.size(); ++i) {
            sortedNums.push_back({nums[i], i});
        }

        sort(sortedNums.begin(), sortedNums.end());

        int left = 0;
        int right = sortedNums.size() - 1;

        while (left < right) {
            int sum = sortedNums[left].first + sortedNums[right].first;
            if (sum == target) {
                return {sortedNums[left].second, sortedNums[right].second};
            } else if (sum < target) {
                ++left;
            } else {
                --right;
            }
        }

        return {};
    }
};
```

```
    }  
};
```

15. 3Sum

layout: post title: 15. 3Sum.md category: leetcode date: 2024-12-18 16:18:45 +0900 description: <https://leetcode.com/problems/3sum/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

15. 3Sum

Given an integer array `nums`, return all the triplets `[nums[i], nums[j], nums[k]]` such that $i \neq j$, $i \neq k$, and $j \neq k$, and $nums[i] + nums[j] + nums[k] == 0$.

Notice that the solution set must not contain duplicate triplets.

```
Example 1:  
Input: nums = [-1,0,1,2,-1,-4]  
Output: [[-1,-1,2],[-1,0,1]]  
Explanation:  
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.  
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.  
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.  
The distinct triplets are [-1,0,1] and [-1,-1,2].
```

Notice that the order of the output and the order of the triplets does not matter.

```
Example 2:  
Input: nums = [0,1,1]  
Output: []  
Explanation: The only possible triplet does not sum up to 0.
```

```
Example 3:  
Input: nums = [0,0,0]  
Output: [[0,0,0]]  
Explanation: The only possible triplet sums up to 0.
```

1차 도전

```

class Solution {
public:
    vector<vector<int>> results;
    vector<vector<int>> threeSum(vector<int>& nums) {
        sort(nums.begin(),nums.end());
        for (int i=1; i<nums.size(); i++){
            int left = i-1;
            int right = i+1;
            while(left < i && i < right && 0<= left && right<nums.size()){
                if (nums[left]+nums[i]+nums[right]==0){
                    vector<int> temp;
                    temp.push_back(nums[left]);
                    temp.push_back(nums[i]);
                    temp.push_back(nums[right]);
                    results.push_back(temp);
                    break;
                }
                else if (nums[left]+nums[i]+nums[right]<0){
                    right++;
                }
                else if (nums[left]+nums[i]+nums[right]>0){
                    left--;
                }
            }
        }
        return results;
    }
};

```

위 코드는 인풋이 아래와같을 때

```
nums = [0,0,0,0]
```

아웃풋이 아래와 같이 나오게 되어 오답.

```
[[0,0,0],[0,0,0]]
```

아웃풋은 중복 없이

```
[[0,0,0]]
```

위와 같이 나와야 함.

2차 도전

```
class Solution {
public:
    vector<vector<int>> results;
    vector<vector<int>> threeSum(vector<int>& nums) {
        sort(nums.begin(),nums.end());
        for (int i=1; i<nums.size(); i++){
            int left = i-1;
            int right = i+1;
            while(left < i && i < right && 0<= left && right<nums.size()){
                if (nums[left]+nums[i]+nums[right]==0){
                    vector<int> temp;
                    temp.push_back(nums[left]);
                    temp.push_back(nums[i]);
                    temp.push_back(nums[right]);
                    results.push_back(temp);
                    left--; //변경된 부분
                }
                else if (nums[left]+nums[i]+nums[right]<0){
                    right++;
                }
                else if (nums[left]+nums[i]+nums[right]>0){
                    left--;
                }
            }
        }
        set<vector<int>> s(results.begin(), results.end());
        vector<vector<int>> results(s.begin(), s.end());
        return results;
    }
};
```

바로 break 했더니 중심은 같은데 좌우가 다른 경우를 못찾았음. 그래서 추가로 탐지 가능하게 변경
하지만 인풋이

[illegible]

위와 같은 경우에는 대처하지 못했음.

3차도전

```
class Solution {
public:
    vector<vector<int>> results;
    vector<vector<int>> threeSum(vector<int>& nums) {
```

부분

```

        sort(nums.begin(),nums.end());
        for (int i=1; i<nums.size(); i++){
            int left = i-1;
            int right = i+1;
            while(left < i && i < right && 0<= left && right<nums.size()){
                if (nums[left]+nums[i]+nums[right]==0){
                    results.push_back({nums[left],nums[i],nums[right]}); //달라진

                    left--;
                }
                else if (nums[left]+nums[i]+nums[right]<0){
                    right++;
                }
                else if (nums[left]+nums[i]+nums[right]>0){
                    left--;
                }
            }
        }
        set<vector<int>> s(results.begin(), results.end());
        vector<vector<int>> results(s.begin(), s.end());
        return results;
    }
};

```

temp를 만들고 다시 results에 넣는 걸 생략해서 겨우 통과할 수 있었음. (시간 효율성은 극악임)

제대로 된 풀이

```

class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> results;
        if (nums.size() < 3) return results;

        sort(nums.begin(), nums.end());

        for (int i = 0; i < nums.size() - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue; //nums[i]중복제거

            int left = i + 1, right = nums.size() - 1; //i,left,right순

            while (left < right) {
                int sum = nums[i] + nums[left] + nums[right];

                if (sum == 0) {
                    results.push_back({nums[i], nums[left], nums[right]});

                    // 중복된 값 피하기
                    while (left < right && nums[left] == nums[left + 1]) left++;
                    while (left < right && nums[right] == nums[right - 1]) right-
                }
            }
        }
    };
}

```



```

        left++;
        right--;
    } else if (sum < 0) {
        left++;
    } else {
        right--;
    }
}
}

return results;
}
};

```

우선 index를 0부터 시작하고, left, right의 시작 위치가 다름.

i를 첫 번째 값으로, left를 중간 값으로, right를 마지막 값으로 이용함.

19. Remove Nth Node From End of List

layout: post title: 19. Remove Nth Node From End of List.md category: leetcode date: 2024-12-18 16:18:45 +0900 description: <https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/> img: leetcode.png # Add image post (optional) figcaption: # Add figcaption (optional)

19. Remove Nth Node From End of List

Given the head of a linked list,

remove the nth node from the end of the list and return its head.

Example 1:

remove_ex1

Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]

Example 2:
Input: head = [1], n = 1
Output: []

Example 3:

Input: head = [1,2], n = 1

Output: [1]

Solution

```
#include <vector>
using namespace std;

//Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x), next(next) {}
};

class Solution {
public:
    int cnt = 0; // 길이를 추적하는 변수
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        if (head == nullptr) {
            return nullptr; // 리스트의 끝에 도달
        }
        head->next = removeNthFromEnd(head->next, n);
        cnt++; // 역순으로 재귀를 돌면서 cnt를 증가
        if (cnt == n) { // 제거할 노드가 현재 노드인 경우
            return head->next; // 현재 노드를 건너뛰
        }
        return head; // 현재 노드 유지
    }
};
```

못풀었던 부분

- cnt를 증가하는걸 역순으로 돌게 하려면 재귀함수 아래에다가 작성했어야함.

추가 참고 사항:

```
if (head == nullptr) {
    return nullptr; // 리스트의 끝에 도달
}
```

이런식으로 쓸 수 있는데 나는

```
if (head == NULL)
```

으로 썼음. 차이가 있나?

그리고,

```
if (cnt == n){  
    return head->next;  
}
```

이렇게 하면 간단하게 가능한데, 나는 좀 복잡하게 풀었음.

```
if (cnt == n){  
    head->next = head->next->next;  
    head->next->next = nullptr;}
```

이런 식으로 풀었는데, Solution 이 더 나은 것 같음.

2. Add Two Numbers

layout: post title: 2. Add Two Numbers.md category: leetcode date: 2024-12-18 16:18:45 +0900 description: <https://leetcode.com/problems/add-two-numbers/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

2. Add Two Numbers

You are given two non-empty linked lists representing two non-negative integers.

The digits are stored in reverse order, and each of their nodes contains a single digit.

Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero,

except the number 0 itself.

 addtwonumber1

Example 1:

Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: 342 + 465 = 807.

Example 2:

Input: l1 = [0], l2 = [0]

Output: [0]

Example 3:

Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

Output: [8,9,9,9,0,0,0,1]

Solution

```
#include <vector>
using namespace std;

//Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x), next(next) {}
};

class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        if (l1 == nullptr && l2 == nullptr) {
            return nullptr;
        }

        // Sum current digits
        int sum = (l1 ? l1->val : 0) + (l2 ? l2->val : 0);

        // Check if next node exists, or if carry should be propagated
        if (l1 && l1->next) {
            l1->next->val += sum / 10; // Add carry to the next node
        } else if (l2 && l2->next) {
```

```
l2->next->val += sum / 10; // Add carry to the next node
} else if (sum >= 10) {
    // Create a new node for carry if no next nodes exist
    if (l1) {
        l1->next = new ListNode(sum / 10);
    } else if (l2) {
        l2->next = new ListNode(sum / 10);
    } else {
        return new ListNode(sum % 10, new ListNode(sum / 10));
    }
}

// Create the current node
ListNode* node = new ListNode(sum % 10);

// Recursively calculate the next node
node->next = addTwoNumbers(l1 ? l1->next : nullptr, l2 ? l2->next :
nullptr);

return node;
}
};
```

문제를 풀다가 마지막 node를 어떻게 할지 고민하다가 시간을 낭비했는데,

맨 위의 생성자 주석 잘 읽으면 간단히 해결 가능한 문제였음. 주의하자.

206. Reverse Linked List

layout: post title: 206. Reverse Linked List.md category: leetcode date: 2024-12-18 16:18:45 +0900 description: <https://leetcode.com/problems/reverse-linked-list/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

206. Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

 rev1ex1

Example 1:
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]

 rev1ex2

Example 2:

Input: head = [1,2]

Output: [2,1]

Example 3:

Input: head = []

Output: []

Solution

```
#include <vector>
using namespace std;

//Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x), next(next) {}
};

class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if (head == nullptr || head->next == nullptr){
            return head;
        }
        ListNode* node = reverseList(head->next);
        head->next->next = head;
        head->next = nullptr;
        return node ;
    }
};
```

고민했던 부분들

1. class 선언하고 사용하는 방법 헛갈림
2. 처음에 빈 노드를 만들고나서 이용하려고 했는데, 오히려 헛갈리게 됨.

3. 결국 이 문제는 node->val 을 채워넣는 건 필요없고, 포인터 방향만 바꾸면되는문제였음

4. 포인터개념 미숙

5.

```
if (head == nullptr || head->next == nullptr)
```

구문을 원래는

```
if ( head->next == nullptr || head == nullptr)
```

로 썼는데,

단락 평가(short-circuit evaluation)로 인해서

```
head->next == nullptr
```

를 컴파일러가 먼저 체크하게 되면 아래와 같이 에러발생함.

```
Line 14: Char 19: runtime error: member access within null pointer of type
'ListNode' (solution.cpp)
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior prog_joined.
```

6. 마지막 head 타고들어갈 때

```
if (head == nullptr || head->next == nullptr){
    return head;
}
```

이용해서 return 해주는걸 깔끔하게 못했음.

Invert Binary Tree

layout: post title: 226. Invert Binary Tree.md category: leetcode date: 2024-12-18 16:18:45 +0900 description: <https://leetcode.com/problems/invert-binary-tree/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

Invert Binary Tree

Given the root of a binary tree, invert the tree, and return its root.

invert1-tree

Example 1:

Input: root = [4,2,7,1,3,6,9]

Output: [4,7,2,9,6,3,1]

invert2-tree'

Example 2:

Input: root = [2,1,3]

Output: [2,3,1]

Example 3:

Input: root = []

Output: []

Solution

```
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
};

class Solution {
public:
    TreeNode* invertTree(TreeNode* root) {
        if(root == NULL) return NULL;
        TreeNode* node = new TreeNode();
        node->val = root->val;
        node->right = invertTree(root->left);
        node->left = invertTree(root->right);
        return node;
    }
};
```



```
};
```

그렇게 어렵지 않으니 잘 생각해보자

4. Median of Two Sorted Arrays

layout: post title: 4. Median of Two Sorted Arrays.md category: leetcode
date: 2024-12-18 16:18:45 +0900 description:
<https://leetcode.com/problems/median-of-two-sorted-arrays/description/> img: leetcode.png # Add image post (optional) fig-
caption: # Add figcaption (optional)

4. Median of Two Sorted Arrays

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively,
return the median of the two sorted arrays.

The overall run time complexity should be $O(\log(m+n))$.

Example 1:

Input: `nums1 = [1,3]`, `nums2 = [2]`

Output: 2.00000

Explanation: merged array = `[1,2,3]` and median is 2.

Example 2:

Input: `nums1 = [1,2]`, `nums2 = [3,4]`

Output: 2.50000

Explanation: merged array = `[1,2,3,4]` and median is $(2 + 3) / 2 = 2.5$.

Binary search

단계

1. 작은 배열에 이진 탐색 수행 보장

- 두 배열 중 작은 배열에서 이진 탐색을 수행하는 것이 효율적입니다. 이는 연산량을 줄이기 위한 것입니다. 따라서 `nums1`의 크기가 `nums2`의 크기보다 크다면 두 배열을 교환합니다.

2. 이진 탐색 초기화

- 이진 탐색의 경계를 low와 high로 설정합니다. 초기값은 low = 0이고, high = nums1.size()입니다.

3. 루프 내에서 이진 탐색 수행

- 이진 탐색은 low가 high보다 작거나 같은 동안 계속됩니다.

4. partitionX와 partitionY 계산

- partitionX는 nums1의 중간 지점을 의미합니다. $(low + high) / 2$ 로 계산됩니다. partitionY는 $(x + y + 1) / 2 - partitionX$ 로 계산됩니다. 이는 전체 길이의 절반에서 partitionX를 뺀 값입니다. 이때, x와 y는 각 배열의 길이입니다.

5. 경계 값 설정

- maxX는 partitionX의 왼쪽 값 중 최대값입니다. partitionX가 0인 경우에는 INT_MIN을 사용합니다. minX는 partitionX의 오른쪽 값 중 최소값입니다. partitionX가 배열의 끝이라면 INT_MAX를 사용합니다. maxY와 minY 역시 동일한 방식으로 partitionY에 대해 설정합니다.

6. 조건 검사

- $maxX \leq minY$ 그리고 $maxY \leq minX$ 인 경우, 올바른 partition을 찾은 것입니다. 이 경우 전체 길이가 짝수인지 홀수인지 판단하여 중간 값을 반환합니다.
 - 짝수인 경우: $(\max(maxX, maxY) + \min(minX, minY)) / 2$
 - 홀수인 경우: $\max(maxX, maxY)$
- $maxX > minY$ 인 경우
 - high를 $partitionX - 1$ 로 업데이트합니다. 이는 partitionX가 너무 오른쪽에 있음을 의미합니다.
- $maxY > minX$ 인 경우
 - low를 $partitionX + 1$ 로 업데이트합니다. 이는 partitionX가 너무 왼쪽에 있음을 의미합니다.

Binary search에서 nums1 num2 교환하는 이유:

이진 탐색 기반 알고리즘의 작동 원리

중간 값을 찾기 위해 이진 탐색을 사용하는 알고리즘은 다음과 같은 절차를 따릅니다.

이진 탐색은 일반적으로 더 작은 배열을 기준으로 수행하여 탐색 구간을 최소화하려고 합니다.

(이진 탐색은 배열의 요소를 절반씩 줄여 나가면서 검색하기 때문에, 더 작은 배열에서 이진 탐색을 수행하는 것이 더 효율적)

작은 배열의 "적절한 위치"를 찾기 위해서 이진 탐색을 수행하여

그 기준으로 반대편 배열에서의 "적절한 위치"를 계산합니다.

이진 탐색을 통해 두 배열에서 합쳐서 "중간"에 위치하는 요소를 찾게 됩니다.

```
#include <vector>
using namespace std;

double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    if (nums1.size() > nums2.size()) {
```

```

        return findMedianSortedArrays(nums2, nums1);
    }

    int x = nums1.size();
    int y = nums2.size();

    int low = 0, high = x;

    while (low <= high) {
        int partitionX = (low + high) / 2;
        int partitionY = (x + y + 1) / 2 - partitionX;

        int maxX = (partitionX == 0) ? INT_MIN : nums1[partitionX - 1];
        int minX = (partitionX == x) ? INT_MAX : nums1[partitionX];

        int maxY = (partitionY == 0) ? INT_MIN : nums2[partitionY - 1];
        int minY = (partitionY == y) ? INT_MAX : nums2[partitionY];

        if (maxX <= minY && maxY <= minX) {
            if ((x + y) % 2 == 0) {
                return (double)(max(maxX, maxY) + min(minX, minY)) / 2;
            } else {
                return (double)max(maxX, maxY);
            }
        } else if (maxX > minY) {
            high = partitionX - 1;
        } else {
            low = partitionX + 1;
        }
    }
}

```

My solution

제한시간 안에는 풀리지만 최적의 방법은 아님.

```
sort(temp.begin(), temp.end());
```

vector sort 하는 방법 순간 잊었었음. 주의 할 것.

```
static_cast(temp[index])
```

type casting 하는 방법도 잊었었음. 주의 할 것.

```

class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        double result = 0;
        int index = 0;
        vector<int> temp;
        for (auto element:nums1){

```

```
        temp.push_back(element);
    }
    for (auto element:nums2){
        temp.push_back(element);
    }
    sort(temp.begin(), temp.end()); // 기억해두자 vectort sort하는 방법 까먹었음
    if (temp.size()%2 ==0){
        index = temp.size()/2 - 1;
        result = (static_cast<double>(temp[index]) + static_cast<double>
(temp[index+1]))/2 ; //이걸까먹음static_cast<double>;
        cout << result << endl;
    }
    else{
        index = temp.size()/2;
        result = temp[index];
    }

    return result;
}
};
```

5. Longest Palindromic Substring

layout: post title: 5. Longest Palindromic Substring.md category: leetcode
date: 2024-12-18 16:18:45 +0900 description:
<https://leetcode.com/problems/longest-palindromic-substring/description/> img: leetcode.png # Add image post (optional)
fig-caption: # Add figcaption (optional)

5. Longest Palindromic Substring

Given a string *s*, return the longest palindromic

substring in *s*.

Example 1:

Input: *s* = "babad" Output: "bab" Explanation: "aba" is also a valid answer. Example 2:

Input: *s* = "cbabd" Output: "bb"

```
class Solution {
public:
    string longestPalindrome(string s) {
        if (s.empty()) return "";

        int start = 0, end = 0;
```

```

// 중간을 기준으로 양쪽을 빼가면서 팰린드롬을 찾음
for (int i = 0; i < s.size(); ++i) { //중심 이동
    int len1 = expandAroundCenter(s, i, i); // 홀수 길이 팰린드롬
    int len2 = expandAroundCenter(s, i, i + 1); // 짝수 길이 팰린드롬
    int len = max(len1, len2);

    if (len > end - start) {
        start = i - (len - 1) / 2;
        end = i + len / 2;
    }
}

return s.substr(start, end - start + 1);
}

private:
int expandAroundCenter(const string& s, int left, int right) {
    while (left >= 0 && right < s.size() && s[left] == s[right]) {
        left--;
        right++;
    }
    return right - left - 1;
}
};

```

기본적으로 좌우 index를 이용해서 좌우를 변경하면서 접근해야함. (2 Pointer)

expandAroundCenter 함수가 핵심인데, 이렇게 깔끔하게 접근하지못해서 실패함.

704. Binary Search

layout: post title: 704. Binary Search.md category: leetcode date: 2024-12-18 16:18:45 +0900 description: <https://leetcode.com/problems/binary-search/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

704. Binary Search

Given an array of integers `nums` which is sorted in ascending order,

and an integer `target`, write a function to search `target` in `nums`.

If `target` exists, then return its index. Otherwise, return -1.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: nums = [-1,0,3,5,9,12], target = 9

Output: 4

Explanation: 9 exists in nums and its index is 4

Example 2:

Input: nums = [-1,0,3,5,9,12], target = 2

Output: -1

Explanation: 2 does not exist in nums so return -1

Solution (Time ↑, Memory ↓)

```
int init = [] {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    ofstream out("user.out");
    cout.rdbuf(out.rdbuf());
    for (string s; getline(cin, s); cout << '\n') {
        string t;
        getline(cin, t);
        int tar = stoi(t);
        for (int i = 0, _i = 1, _n = s.length(); _i < _n; ++i, ++_i) {
            bool _neg = 0;
            if (s[_i] == '-')
                ++_i, _neg = 1;
            int v = s[_i++] & 15;
            while ((s[_i] & 15) < 10)
                v = v * 10 + (s[_i++] & 15);
            if (_neg)
                v = -v;
            if (v == tar) {
                cout << i;
                goto next;
            }
            if (v > tar)
                break;
        }
        cout << -1;
    next:;
    }
    exit(0);
    return 0;
}();
```

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int first = 0, last = nums.size() - 1;

        while (first <= last) {
            int mid = first + (last - first) / 2;
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                first = mid + 1; // Move to the right half
            } else {
                last = mid - 1; // Move to the left half
            }
        }
        return -1; // Target not found
    }
};
```

이 방식은 4ms 가 걸렸지만, Memory는 아래의 내 코드보다 4배 적게 사용함. (8MB)

상세한 내용은 찾아보자

My Solution

```
#include <vector>
using namespace std;
class Solution
{
public:
    int index = 0;
    int result = -1;
    int temp = 0;

    int search(vector<int>& nums, int target)
    {
        if (nums.empty()) // nums가 비어있는지 확인
        {
            return -1;
        }

        index = nums.size() / 2;

        if (target == nums[index]) // 타겟 숫자와 중간 값이 같을 때
        {
            return index + temp;
        }

        if (target > nums[index]) // 타겟 숫자가 중간 값보다 클 때
        {
```

```
        vector<int> sliced_nums(nums.begin() + index + 1, nums.end());
        temp += index + 1;
        return search(sliced_nums, target);
    }

    // 타겟 숫자가 중간 값보다 작을 때
    vector<int> sliced_nums(nums.begin(), nums.begin() + index);
    return search(sliced_nums, target);
}
};
```

원리를 알고 있으면 어려운 문제는 아니지만,

index 사용에 있어서 실수가 많아서 Submit 을 여러 번 함.

94. Binary Tree Inorder Traversal

layout: post title: 94. Binary Tree Inorder Traversal.md category: leetcode
date: 2024-12-18 16:18:45 +0900 description:
<https://leetcode.com/problems/binary-tree-inorder-traversal/description/>
img: leetcode.png # Add image post (optional) fig-caption: # Add
figcaption (optional)

94. Binary Tree Inorder Traversal

Given the root of a binary tree, return the inorder traversal of its nodes' values.


Example 1:
Input: root = [1,null,2,3]
Output: [1,3,2]

Explanation:

 Binary_Tree_Inorder_Traversal_1

Example 2:
Input: root = [1,2,3,4,5,null,8,null,null,6,7,9]
Output: [4,2,6,5,7,1,3,9,8]

Explanation:

 Binary_Tree_Inorder_Traversal_2

Example 3:
 Input: root = []
 Output: []

Example 4:
 Input: root = [1]
 Output: [1]

중위 순회 문제

풀었었던 문제인데 틀려버렸다....

답안

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    void inorderTraversalHelper(TreeNode* root, vector<int>& result) {
        if (root == nullptr) {
            return;
        }
        inorderTraversalHelper(root->left, result); //좌측 순회부터
        result.push_back(root->val); //중간에 집어넣어야 좌측->root->우측
        inorderTraversalHelper(root->right, result);
        //중위순회를 위한 로직.
    }

    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorderTraversalHelper(root, result);
        return result;
    }
};
```

틀린 원인

1. 갑자기 class 객체 생성 방법이 안떠오름
2. 파라미터 확인을 제대로 못했음.
3. 문제 이해를 제대로 못함

98. Validate Binary Search Tree

layout: post title: 98. Validate Binary Search Tree.md category: leetcode
date: 2024-12-18 16:18:45 +0900 description:
<https://leetcode.com/problems/validate-binary-search-tree/description/>
img: leetcode.png # Add image post (optional) fig-caption: # Add
figcaption (optional)

98. Validate Binary Search Tree


Binary Search Tree

Given the root of a binary tree, determine if it is a valid binary search tree (BST).

A valid BST is defined as follows:

The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. Both the left and right subtrees must also be binary search trees.

Example 1:

 Validate Binary Search Tree_1

```
Input: root = [2,1,3]
Output: true
```

Example 2:

 Validate Binary Search Tree_2

```
Input: root = [5,1,4,null,null,3,6]
Output: false
Explanation: The root node's value is 5 but its right child's value is 4.
```

1차 도전

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
```

```

*   int val;
*   TreeNode *left;
*   TreeNode *right;
*   TreeNode() : val(0), left(nullptr), right(nullptr) {}
*   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
*   TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
* };
*/
class Solution {
public:
    bool dfs(TreeNode* root){
        bool temp = true;

        if (root==nullptr){
            return true;
        }

        if (root->left != nullptr && root->left->val >= root->val){
            return false;
        }
        if (root->right != nullptr && root->right->val <= root->val){

            return false;
        }
        temp = dfs(root->left);
        temp = dfs(root->right);
        return temp;
    }
    bool isValidBST(TreeNode* root) {
        bool result = dfs(root);
        return result;
    }
};

```

이 풀이의 문제점은

```
root = [5,4,6,null,null,3,7]
```

일 때, 에러가 발생함.

이유는 subtree에서도 최상단 root 와 value비교를 해야한다는거..

6의 left인 3이 root->val 인 5 보다 작기 때문에 false가 나와야한다.

이 코드의 문제점은,

아래에서부터 올라가는 경우에는 부모 노드의 값을 저장해놓을수가 없으니 비교하기가 힘들다.

해결책

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    bool dfs(TreeNode* root, TreeNode* minNode = nullptr, TreeNode* maxNode =
nullptr) {
        if (root == nullptr) {
            return true;
        }
        if ((minNode != nullptr && root->val <= minNode->val) || (maxNode !=
nullptr && root->val >= maxNode->val)) {
            return false;
        }
        //깔끔해진 조건문 처리 기억하자
        return dfs(root->left, minNode, root) && dfs(root->right, root, maxNode);
        //return 을 바로 dfs && 로 깔끔하게,
    }

    bool isValidBST(TreeNode* root) {
        return dfs(root);
    }
};

```

검사를 할 때, 자식노드가 아닌 부모 노드를 체크하면서 올라가는법을 이용해야함.

우측에서 올라갈때는 우측 비교 노드 (maxNode) 를 nullptr로 만들어서 비교 및 이용하고,

좌측에서 올라갈때는 좌측 비교 노드 (minNode)를 nullptr로 만들어서 이용한다.

99. Recover Binary Search Tree


layout: post title: 99. Recover Binary Search Tree.md category: leetcode
date: 2024-12-18 16:18:45 +0900 description:
<https://leetcode.com/problems/recover-binary-search-tree/description/>
img: leetcode.png # Add image post (optional) fig-caption: # Add
figcaption (optional)

99. Recover Binary Search Tree

Binary Search Tree


You are given the root of a binary search tree (BST), where the values of exactly two nodes of the tree were swapped by mistake. Recover the tree without changing its structure.

Example 1:

 Validate Binary Search Tree_1

```
Input: root = [1,3,null,null,2]
Output: [3,1,null,null,2]
Explanation: 3 cannot be a left child of 1 because 3 > 1. Swapping 1 and 3 makes the BST valid.
```

Example 2:

 Validate Binary Search Tree_2

```
Input: root = [3,1,4,null,null,2]
Output: [2,1,4,null,null,3]
Explanation: 2 cannot be in the right subtree of 3 because 2 < 3. Swapping 2 and 3 makes the BST valid.
```

풀이 실패

내 코드

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    void myfunction(TreeNode* node,vector<int>& results,TreeNode*minNode,
TreeNode* maxNode){
        if(node==nullptr){
            cout <<"0"<<endl;
            return;
        }
        myfunction(node->left,results,nullptr,node);
```

```

myfunction(node->right,results,node,nullptr);
if(minNode!=nullptr && node->val < minNode->val){
    int temp = minNode->val;
    minNode->val = node->val;
    node->val = temp;
}
if(maxNode!=nullptr && node->val > maxNode->val){
    int temp = maxNode->val;
    maxNode->val = node->val;
    node->val = temp;
}
results.push_back(node->val);
return;
}

void recoverTree(TreeNode* root) {
    vector<int> results;
    myfunction(root,results,nullptr,nullptr);

}

};

```

잘못된 부분 설명

1. 중위 순회를 제대로 구현하지 못함

중위 순회는 왼쪽 노드 -> 현재 노드 -> 오른쪽 노드 순이어야 하지만,
 기존 함수는 왼쪽과 오른쪽 자식 노드를 오가며 잘못된 위치 검사만 수행
 → 중위 순회를 할 때에는 꼭

```

inorder(root->left,...);
//추가 코드//
inorder(root->right,...);

```

위의 형태를 지키도록 하자.

2. 부모와 자식 1세대만 가지고는 풀 수가 없다.

기본적으로 이 문제 같은 경우에는 subtree까지 전부 확인해야하므로,
 단순히 1세대비교만으로는 구조적으로 불가능하다.
 때문에, 3개의 TreeNode ptr을 만들어 2세대간 비교를 할 수 있게 해야한다.

풀이법

1. Three Pointer 풀이법

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    TreeNode* first = nullptr;
    TreeNode* second = nullptr;
    TreeNode* prevNode = nullptr;

    void myfunction(TreeNode* node) {
        if (node == nullptr) {
            return;
        }
        myfunction(node->left);
        if (prevNode != nullptr && node->val < prevNode->val) {
            if (first == nullptr) {
                first = prevNode;
            }
            second = node;
        }
        prevNode = node;
        myfunction(node->right);
    }

    void recoverTree(TreeNode* root) {
        myfunction(root);
        if (first != nullptr && second != nullptr) {
            int temp = first->val;
            first->val = second->val;
            second->val = temp;
        }
    }
};

```

98. Validate Binary Search Tree 문제의 경우

Pointer가 추가적으로 2개면 충분했다. (교환이 아닌 검증이기 때문에)

그러나 이 경우에는 교환작업이 필요하므로, 포인터가 3개 필요하다.

prev 포인터

- 역할 : 중위 순회를 하면서 이전 노드를 추적합니다.
- 이유 : 이진 검색 트리(BST)에서 중위 순회는 값이 오름차순으로 되기 때문에, 현재 노드의 값은 이전 노드(prev)의 값보다 커야 합니다. 이를 통해 현재 노드와 이전 노드를 비교하여 교환이 필요한 노드를 탐지할 수 있습니다.

first 포인터

- 역할 : 잘못된 노드쌍 중 첫 번째 노드를 저장합니다.
- 이유 : 최초로 발견된 순서가 잘못된 노드(prev > 현재 노드)를 가리킵니다. 이 노드는 교체 대상 중 하나입니다. 여러 번 발견될 수도 있지만, 우리는 처음 발견된 노드를 저장합니다.

second 포인터

- 역할 : 잘못된 노드쌍 중 두 번째 노드를 저장합니다.
- 이유 : 이후에 발견된 순서가 잘못된 노드(prev > 현재 노드)를 가리킵니다. 이 노드는 두 번째 교체 대상입니다. 만약 여러 번 발견되더라도 마지막에 발견된 노드를 저장합니다.

21. Merge Two Sorted Lists

layout: post title: 21. Merge Two Sorted Lists category: leetcode date: 2025-01-10 12:45:00 +0900 description: <https://leetcode.com/problems/merge-two-sorted-lists/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

21. Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

```
Example 1:
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

```
Example 2:
Input: list1 = [], list2 = []
Output: []
```


Example 3:
 Input: list1 = [], list2 = [0]
 Output: [0]

Solution (Failed)

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        if (list1 && list2){
            ListNode * node = new ListNode ();
            ListNode * temp = new ListNode ();
            if(list1->val < list2-> val){
                int k= list1->val;
                list1->val = list2->val;
                list2->val = k;
            }

            temp->val = list2->val;
            temp->next = list2->next;
            list2->next = list1;
            mergeTwoLists(temp->next, list1->next);
        }
        else if (!list1 && list2){
            return list2;
        }
        else if (list1 && !list2){
            return list1;
        }
        return list2;
    }
};
```

난관 1. 오랜만에 해서

```
ListNode * node = new ListNode ();
```

이런 식으로 동적할당 해주는 방법을 잊어버림;;;

난관 2. 문제를 잘못 이해해서, 같은 사이즈의 node가 들어온다고 가정하고 풀어서 Exception에 많이 걸렸음.

난관 3. 문제에서 Sorted 부분을 읽지못하고 그냥 붙여주기만했음

난관 4. sorting을 단순히 두 값 비교만해서 넣다보니, 순서가 꼬여서 오답이 발생함.

Solution

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        if (list1 == nullptr) {
            return list2;
        } else if (list2 == nullptr) {
            return list1;
        }
        // 여기 아래만 보면 됨
        // 현재 값 비교 후 재귀 호출을 통해 병합
        if (list1->val < list2->val) {
            list1->next = mergeTwoLists(list1->next, list2);
            return list1;
        } else {
            list2->next = mergeTwoLists(list1, list2->next);
            return list2;
        }
    }
};
```

재귀를 그대로 둔 상태로 푼 풀이.

(More stack memory required) (스택메모리 요구량이 많다고해서 더 빠른건 아니다.)

Solution (While loop)

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {

        ListNode dummy;
        ListNode* tail = &dummy;

        while (list1 != nullptr && list2 != nullptr) {
            if (list1->val < list2->val) {
                tail->next = list1;
                list1 = list1->next;
            } else {
                tail->next = list2;
                list2 = list2->next;
            }
            tail = tail->next;
        }

        if (list1 != nullptr) {
            tail->next = list1;
        }
    }
};
```

```
    } else {  
        tail->next = list2;  
    }  
  
    return dummy.next;  
}  
};
```

오늘의 교훈 :

문제를 끝까지 제대로 읽자.

읽는데 시간이 더 걸리더라도 정확하게 푸는게 중요

70. Climbing Stairs

layout: post title: 70. Climbing Stairs category: leetcode date: 2025-01-14 13:58:00 +0900 description: <https://leetcode.com/problems/climbing-stairs/description/> img: leetcode.png # Add image post (optional) figcaption: # Add figcaption (optional)

70. Climbing Stairs

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?

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

문제 자체는 아주 쉬운 피보나치 수열 문제지만, 나는 고민했다.

일단, 고등학교 수학 계산하듯이 풀어도되나 싶어서...(수식 생성)

Solution (Failed)

```
class Solution {
public:
    int climbStairs(int n) {
        if (n==0){
            return 0;
        }
        if (n==1){
            return 1;
        }
        if (n==2){
            return 2;
        }
        return climbStairs(n-1) + climbStairs(n-2);
    }
};
```

이 풀이는 수학적으로는 맞을 수 있는데,

결론적으로 말하자면 실패한 풀이이다.

재귀적으로 함수를 호출한다는 것은

1. 스택 프레임 할당

- 스택 프레임에는 함수의 매개변수, 지역 변수, 반환 주소 등이 저장.

2. 매개변수 전달

3. 레지스터 저장

4. 제어 흐름 변경

- 함수 호출은 현재 실행 흐름을 중단하고 새로운 함수로 이동하게 합니다. 호출된 함수의 실행이 끝나면 다시 원래의 호출 지점으로 돌아와야 하므로, 이에 따른 제어 흐름 변경 비용이 발생.

5. 캐시 미스

- 재귀 호출이 이루어질 때, 특히 깊은 재귀 호출의 경우, 메모리 캐시 미스와 같은 추가적인 메모리 관리 비용이 발생할 수 있음.

위의 5가지 작업이 동반되는데, 재귀적으로 하면 계속해서 함수호출이 일어나므로

메모리, 시간적 낭비가 발생한다.

시간복잡도 : $O(2^n)$

그래서 $n=45$ 일때 Error가 발생함.

이를 해결하기 위한 방법이 바로 DP 이다.

Solution

```

class Solution {
public:
    vector<int> v;
    int climbStairs(int n) {
        v.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            if (i == 0) {
                v[i] = 0;
            } else if (i == 1) {
                v[i] = 1;
            } else if (i == 2) {
                v[i] = 2;
            } else {
                v[i] = v[i - 1] + v[i - 2];
            }
        }

        return v[n];
    }
};

```

우선, resize 나 assign 이용해서 n+1만큼 vector공간 마련해주기.

실수 한 부분

1. assign이나 resize를 안하고 push_back 만 하다가 오답냈음
2. else if 안하고 if만 했다가 오류 발생 (멍청한 실수....)

9. Palindrome Number

layout: post title: 9. Palindrome Number.md category: leetcode date: 2025-01-10 09:29:00 +0900 description: <https://leetcode.com/problems/palindrome-number/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

9. Palindrome Number

Given an integer x, return true if x is a palindrome , and false otherwise.

Example 1:

Input: x = 121 Output: true Explanation: 121 reads as 121 from left to right and from right to left. Example 2:

Input: x = -121 Output: false Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome. Example 3:

Input: x = 10 Output: false Explanation: Reads 01 from right to left. Therefore it is not a palindrome.

Solution

```
class Solution {
public:
    bool isPalindrome(int x) {
        if(x<0 || (x!=0 && x%10==0)) return false;
        int sum=0;
        while(x>sum)
        {
            sum = sum*10+x%10;
            x = x/10;
        }
        return (x==sum) || (x==sum/10);
    }
};
```

어려운 문제가 아니라서 인터넷의 좋은 예시를 보고 개선하기로 했다.

아이디어는 나와 같았는데,

```
sum = (sum * 10) + (x % 10);
x = x / 10;
```

이런 식으로 역순 숫자 sum 을만들어서 진행하는 점이 인상깊었음.

```
return (x==sum) || (x==sum/10);
```

이 부분도 좋은 것 같다.

18. 4Sum

layout: post title: 18. 4Sum category: leetcode date: 2025-01-17 01:31:00 +0900 description: <https://leetcode.com/problems/4sum/description/> envType=problem-list-v2&envId=25uoksw3 img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

18. 4Sum

Given an array `nums` of `n` integers, return an array of all the unique quadruplets `[nums[a], nums[b], nums[c], nums[d]]` such that:

$0 \leq a, b, c, d < n$, a, b, c , and d are distinct. $nums[a] + nums[b] + nums[c] + nums[d] == target$ You may return the answer in any order.

Example 1:

Input: `nums = [1,0,-1,0,-2,2]`, `target = 0`

Output: `[[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]`

Example 2:

Input: `nums = [2,2,2,2,2]`, `target = 8`

Output: `[[2,2,2,2]]`

Leetcode에서 본 문제 중 가장 쓰레기같은 문제.

함정이 있는 문제인줄 알고 한참 고민했는데 너무 짜증남

Solution

```
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        sort(nums.begin(), nums.end());
        vector<vector<int>> result;

        for (int i = 0; i < nums.size(); i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue;

            for (int j = i + 1; j < nums.size(); j++) {
                if (j > i + 1 && nums[j] == nums[j - 1]) continue;

                int left = j + 1, right = nums.size() - 1;
                while (left < right) {
                    long long sum = (long long)nums[i] + nums[j] + nums[left] +
nums[right];

                    if (sum == target) {
                        result.push_back({nums[i], nums[j], nums[left],
nums[right]});

                        while (left < right && nums[left] == nums[left + 1])
left++;

                        while (left < right && nums[right] == nums[right - 1])
right--;

                        left++;
                    }
                }
            }
        }

        return result;
    }
};
```

```

        right--;
    } else if (sum < target) {
        left++;
    } else {
        right--;
    }
}
}
}
return result;
}
};

```

풀이가 의미가 없다.

그냥 $O(n^3)$ 짜리 문제.

이런건 물어보지 않을 것 같다...

23. Merge k Sorted Lists

layout: post title: 23. Merge k Sorted Lists category: leetcode date: 2025-01-17 01:31:00 +0900 description:
<https://leetcode.com/problems/merge-k-sorted-lists/description/?envType=problem-list-v2&envId=25uoksw3> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

23. Merge k Sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:
 Input: lists = [[1,4,5],[1,3,4],[2,6]]
 Output: [1,1,2,3,4,4,5,6]
 Explanation: The linked-lists are:
 [
 1->4->5,
 1->3->4,
 2->6
]
 merging them into one sorted list:
 1->1->2->3->4->4->5->6

Example 2:
 Input: lists = []
 Output: []

Example 3:
 Input: lists = [[]]
 Output: []

Solution

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    vector<int> v;

    ListNode* mergeKLists(vector<ListNode*>& lists) {
        for(int i=0; i<lists.size(); i++){
            while(lists[i]){
                v.push_back(lists[i]->val);
                lists[i] = lists[i]->next;
            }
        }
        sort(v.begin(),v.end());

        ListNode* dummy = new ListNode();
        ListNode* current = dummy;

        for (int i = 0; i < v.size(); i++) {
            current->next = new ListNode(v[i]);
            current = current->next;
        }

        ListNode* sortedList = dummy->next;

        return sortedList;
    }
};
```

생각보다 쉬운 문제여서 오히려 함정인가 생각해서 오래걸림.

그냥 전부 받아와서 벡터에 넣고, 정렬후에 다시 ListNode로 만들었음.

시간 가장 많이 쓴 부분은

```
ListNode* dummy = new ListNode();
ListNode* current = dummy;

for (int i = 0; i < v.size(); i++) {
    current->next = new ListNode(v[i]);
    current = current->next;
}

ListNode* sortedList = dummy->next;
```

순간 반복문 통해서 연속적으로 연결해주는 방법이 기억안났음. 주의하자.

2529. Maximum Count of Positive Integer and Negative Integer

layout: post title: 2529. Maximum Count of Positive Integer and Negative Integer category: leetcode date: 2025-01-18 04:49:00 +0900 description: <https://leetcode.com/problems/maximum-count-of-positive-integer-and-negative-integer/description/?envType=problem-list-v2&envId=binary-search> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

2529. Maximum Count of Positive Integer and Negative Integer

Given an array `nums` sorted in non-decreasing order, return the maximum between the number of positive integers and the number of negative integers.

In other words, if the number of positive integers in `nums` is `pos` and the number of negative integers is `neg`, then return the maximum of `pos` and `neg`. Note that 0 is neither positive nor negative.

Example 1:

Input: `nums = [-2, -1, -1, 1, 2, 3]`

Output: 3

Explanation: There are 3 positive integers and 3 negative integers. The maximum count among them is 3.

Example 2:

Input: nums = [-3,-2,-1,0,0,1,2]

Output: 3

Explanation: There are 2 positive integers and 3 negative integers. The maximum count among them is 3.

Example 3:

Input: nums = [5,20,66,1314]

Output: 4

Explanation: There are 4 positive integers and 0 negative integers. The maximum count among them is 4.

Solution

```
class Solution {
public:
    int neg_cnt = 0;
    int pos_cnt = 0;
    int maximumCount(vector<int>& nums) {
        for(int i=0; i<nums.size(); i++){
            if(nums[i]<0){
                neg_cnt ++;
            }
            else if(nums[i]>0){
                pos_cnt ++;
            }
        }
        return (pos_cnt > neg_cnt) ? pos_cnt : neg_cnt;
    }
};
```

너무 쉬워서 설명 생략

2540. Minimum Common Value

layout: post title: 2540. Minimum Common Value category: leetcode date: 2025-01-18 04:49:00 +0900 description:

<https://leetcode.com/problems/minimum-common-value/description/?envType=problem-list-v2&envId=binary-search> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

2540. Minimum Common Value

Given two integer arrays `nums1` and `nums2`, sorted in non-decreasing order, return the minimum integer common to both arrays. If there is no common integer amongst `nums1` and `nums2`, return `-1`.

Note that an integer is said to be common to `nums1` and `nums2` if both arrays have at least one occurrence of that integer.

Example 1:

Input: `nums1 = [1,2,3]`, `nums2 = [2,4]`

Output: 2

Explanation: The smallest element common to both arrays is 2, so we return 2.

Example 2:

Input: `nums1 = [1,2,3,6]`, `nums2 = [2,3,4,5]`

Output: 2

Explanation: There are two common elements in the array 2 and 3 out of which 2 is the smallest, so 2 is returned.

Solution (Hash table)

```
class Solution {
public:
    int getCommon(vector<int>& nums1, vector<int>& nums2) {
        unordered_map<int, int> map;
        for (int num : nums1) {
            map[num] = 1;
        }
        bool found = false;
        int minCommon = INT_MAX;
        for (int num : nums2) {
            if (map.find(num) != map.end()) {
                minCommon = min(minCommon, num);
                found = true;
            }
        }
        return found ? minCommon : -1;
    }
};
```

쉬운 문제인데 적은 이유 : Hash table을 이용한 접근과 Two pointer를 이용한 접근의 차이 확인

Hash 이용할 경우, Hash table에 삽입을 하는데 $O(n)$ 의 시간 소요

```
for (int num : nums1) {
    map[num] = 1;
}
```

```
}
```

탐색에 $O(1)$, 전체 탐색 비용 $O(m)$ 즉, $O(n+m)$ 인데, 사실 Two pointer를 사용하더라도 시간복잡도는 같다.

그러나 이 문제의 경우 정렬된 배열이었으므로, 정렬에 드는 비용이 Two pointer에서 들지 않았기 때문에 유리하다.

만약 정렬해야하는 과정이 있었다면 $O(n\log n + m\log m)$ 이 추가로 들었을 것.

Solution 2 (Two pointer)

```
class Solution {
public:
    int getCommon(vector<int>& nums1, vector<int>& nums2) {
        int i = 0, j = 0;
        // Two-pointer approach
        while (i < nums1.size() && j < nums2.size()) {
            if (nums1[i] == nums2[j]) {
                return nums1[i];
            } else if (nums1[i] < nums2[j]) {
                i++;
            } else {
                j++;
            }
        }

        return -1;
    }
};
```

Two pointer 는 그냥 인덱스 쓰는거라 메모리 추가사용 없음

35. Search Insert Position

layout: post title: 35. Search Insert Position category: leetcode date: 2025-01-17 01:31:00 +0900 description: <https://leetcode.com/problems/search-insert-position/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

35. Search Insert Position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: nums = [1,3,5,6], target = 5

Output: 2

Example 2:

Input: nums = [1,3,5,6], target = 2

Output: 1

Example 3:

Input: nums = [1,3,5,6], target = 7

Output: 4

Solution

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        for(int i=0; i<nums.size(); i++){
            if(target == nums[i]){
                return i;
            }
            if(target < nums[i]){
                return i;
            }
            else if (target > nums[i] && i==nums.size()-1){
                return i+1;
            }
        }
        return 0;
    }
};
```

876. Middle of the Linked List

layout: post title: 876. Middle of the Linked List category: leetcode date: 2025-01-18 08:25:00 +0900 description: <https://leetcode.com/problems/middle-of-the-linked-list/> img:

leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

876. Middle of the Linked List

Given the head of a singly linked list, return the middle node of the linked list.

If there are two middle nodes, return the second middle node.

Example 1:
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.

Example 2:
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

Solution

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    int index = 0;
    ListNode* middleNode(ListNode* head) {
        ListNode* node = head;
        while(head){
            index++;
            head = head->next;
        }
        for(int i=0; i<index/2; i++){
            node = node->next;
        }
        return node;
    }
};
```

```
}  
};
```

딱히 설명할게 없음

Solution 2

```
class Solution {  
public:  
    ListNode* middleNode(ListNode* head) {  
        ListNode* slow = head;  
        ListNode* fast = head;  
  
        while (fast != nullptr && fast->next != nullptr) {  
            slow = slow->next;  
            fast = fast->next->next; // 빠른 포인터는 두 칸 이동  
        }  
  
        return slow;  
    }  
};
```

이렇게 구성해주면 시간복잡도가 $O(1.5n)$ 에서 $O(n)$ 이 됨.

14. Longest Common Prefix

layout: post title: 14. Longest Common Prefix category: leetcode date: 2025-01-21 11:09:00 +0900 description: <https://leetcode.com/problems/longest-common-prefix/description/?envType=problem-list-v2&envId=25uoksw3> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

14. Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

Example 1:
Input: strs = ["flower", "flow", "flight"]
Output: "fl"

Example 2:
 Input: strs = ["dog","racecar","car"]
 Output: ""
 Explanation: There is no common prefix among the input strings.

진짜 쉬운 문제인데 문자열에 익숙하지 않아서 틀림.

Solution

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if (strs.empty()) return "";
        sort(strs.begin(), strs.end());
        string first = strs[0];
        string last = strs[strs.size() - 1];
        int i = 0;
        while (i < first.size() && first[i] == last[i]) {
            i++;
        }
        return first.substr(0, i);
    }
};
```

sort가 가장 핵심인데, sort를 하면 어차피 사전순 정렬이 된다.

처음과 마지막만 비교하는 이유가, 어차피 사전정렬하면 처음과 마지막의 공통 prefix가 어차피中间的 애들과 겹치므로 상관없음.

20. Valid Parentheses

layout: post title: 20. Valid Parentheses category: leetcode date: 2025-01-21 16:40:00 +0900 description: <https://leetcode.com/problems/valid-parentheses/description/?envType=problem-list-v2&envId=25uoksw3> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

20. Valid Parentheses

Given a string *s* containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets. Open brackets must be closed in the correct order. Every close bracket has a corresponding open bracket of the same type.

Example 1:
Input: s = "()"
Output: true

Example 2:
Input: s = "()[]{}"
Output: true

Example 3:
Input: s = "]"
Output: false

Example 4:
Input: s = "[]"
Output: true

Solution

```
class Solution {
public:
    bool isValid(string s) {
        unordered_map<char, char> um = {{'}', '{'}, {'}', '('}, {'}', '['}, {'}', '['}};
        stack<char> stack;
        for (char& c : s) {
            if (um.find(c) != um.end()) {
                //find함수는 찾고자하는 타겟이 없으면 .end()를 반환한다.
                if (!stack.empty() && stack.top() == um[c]) {
                    stack.pop();
                } else {
                    return false;
                }
            } else {
                stack.push(c);
            }
        }
        return stack.empty();
    }
};
```

26. Remove Duplicates from Sorted Array

layout: post title: 26. Remove Duplicates from Sorted Array category: leetcode date: 2025-01-21 17:12:00 +0900 description: <https://leetcode.com/problems/remove-duplicates-from-sorted-array/description/?envType=problem-list-v2&envId=25uoksw3> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

26. Remove Duplicates from Sorted Array

Given an integer array `nums` sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in `nums`.

Consider the number of unique elements of `nums` to be `k`, to get accepted, you need to do the following things:

Change the array `nums` such that the first `k` elements of `nums` contain the unique elements in the order they were present in `nums` initially. The remaining elements of `nums` are not important as well as the size of `nums`. Return `k`. Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) { assert nums[i] == expectedNums[i]; }
```

If all assertions pass, then your solution will be accepted.

Example 1:

Input: `nums = [1,1,2]`

Output: 2, `nums = [1,2,_]`

Explanation: Your function should return `k = 2`, with the first two elements of `nums` being 1 and 2 respectively.

It does not matter what you leave beyond the returned `k` (hence they are underscores).

Example 2:

Input: `nums = [0,0,1,1,1,2,2,3,3,4]`

Output: 5, `nums = [0,1,2,3,4,_,_,_,_,_]`

Explanation: Your function should return `k = 5`, with the first five elements of `nums` being 0, 1, 2, 3, and 4 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

for loop 시작, 끝지점 주의할 것

Solution

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if (nums.size() == 0) return 0;
        int index = 1;

        for (int i = 0; i < nums.size(); ++i) {
            if (i == 0 || nums[i] != nums[i - 1]) {
                nums[index - 1] = nums[i];
                index++;
            }
        }
        return index - 1;
    }
};
```

39. Combination Sum

layout: post title: 39. Combination Sum category: leetcode date: 2025-01-21 14:50:00 +0900 description: img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

39. Combination Sum

Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order.

The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:
Input: candidates = [2,3,6,7], target = 7

Output: `[[2,2,3],[7]]`

Explanation:

2 and 3 are candidates, and $2 + 2 + 3 = 7$. Note that 2 can be used multiple times.
7 is a candidate, and $7 = 7$.

These are the only two combinations.

Example 2:

Input: candidates = `[2,3,5]`, target = 8

Output: `[[2,2,2,2],[2,3,3],[3,5]]`

Example 3:

Input: candidates = `[2]`, target = 1

Output: `[]`

시간초과로 Fail, 중복회피하기 위한 start 지점 이용을 생각못함.

Solution

```
class Solution {
public:
    int temp = 0;
    vector<vector<int>> results;
    vector<int> current;

    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        calculation(candidates, target, 0);
        return results;
    }

    void calculation(vector<int>& candidates, int target, int start) {
        if (target == 0) {
            results.push_back(current);
            return;
        }

        for (int i = start; i < candidates.size(); i++) {
            if (candidates[i] <= target) {
                current.push_back(candidates[i]);
                temp = target - candidates[i];
                calculation(candidates, temp, i); // `i`를 전달하여 중복 조합을 방지
                current.pop_back();
            }
        }
    }
};
```

46. Permutations

layout: post title: 46. Permutations category: leetcode date: 2025-01-21 11:09:00 +0900 description: <https://leetcode.com/problems/permutations/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

46. Permutations

Given an array `nums` of distinct integers, return all the possible permutations . You can return the answer in any order.

Example 1:
Input: `nums = [1,2,3]`
Output: `[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]`

Example 2:
Input: `nums = [0,1]`
Output: `[[0,1],[1,0]]`

Example 3:
Input: `nums = [1]`
Output: `[[1]]`

풀이 실패, `next_permutation` 쓰지않는 조건으로 시도.

Solution

```
class Solution {
public:
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> results;
        vector<int> current;
        vector<bool> used(nums.size(), false);
        backtrack(nums, results, current, used);
        return results;
    }
};
```

```
private:
    void backtrack(vector<int>& nums, vector<vector<int>>& results, vector<int>&
current, vector<bool>& used) {
        if (current.size() == nums.size()) {
            results.push_back(current);
            return;
        }

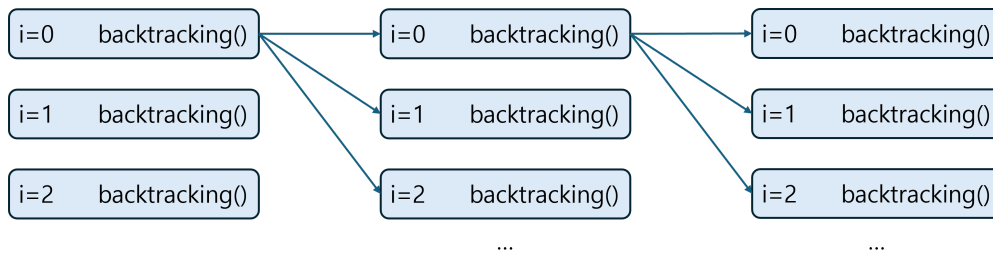
        for (int i = 0; i < nums.size(); ++i) {
            if (used[i]) continue;
            used[i] = true;
            current.push_back(nums[i]);
            backtrack(nums, results, current, used);
            current.pop_back();
            used[i] = false;
        }
    }
};
```

DFS를 이용한 풀이. used라는 vector를 이용해서 backtracking 해준다.

backtrack() 이후 current.pop_back()을 통해서.

backtracking을 하기 위한 준비 단계로

최종 결과를 저장할 results, 현재 상태를 기록할 current, 방문 확인을 위한 used 가 쓰인다.



대략적인 방식은 위 그림과 같다.

일단, 가능한 모든 경우의수에 접근하되,

불필요한 부분은 used를 이용하여 접근제한을 걸어 만든다.

vector nums = {1, 2, 3}; 라고 할 때,

for loop를 처음 타면

i=0 일때 (최초 방문), used[0] = true로 만들고 current에 nums[0]을 넣는다.

그 다음 backtrack()을 처음으로 호출하게 되고,

다시 for loop를 **i=0** 으로 방문하는데, used[0] 가 이미 true로 설정되어있으므로, continue

i=1 이 되면 used[1] = true로 바꾸고, current에 nums[1]추가.

이후 backtrack()호출. 마찬가지로 i=0, i=1일때는 continue

i=2 일때 `used[2] = true` 로 바꾼 뒤 `nums[2]` 삽입.

이후 `backtrack()`호출하는데, 이제 `nums`와 `current`의 길이가 같으므로 `return`

여기서, `return`되었으므로 바로 아래의 `current.pop_back()`; 이 호출된다.

3이 삭제되고, `used[2]` 를 `false`로 바꾼다.

재귀 호출되었으므로, `backtrack()`이 현재 3번호출되었기에 `pop_back()`이 2번이루어진다.

(첫번째 `backtrack()`는 `permute`에서 호출한거임.)

그러면 `current`는 `[1]`이 된다.

중요한 점은,

`backtrack`이 2번째 호출될때의 `i`는 1이고,

세 번째 호출될 때의 `i`는 2이므로, `pop_back()`이 끝나고 `current`는 `[1]`일때,

`i`값은 1이라는 뜻이다.

`for loop`의 조건에 의해서 `i=2`가되어 다시 `for loop`에 진입하고,

현재 `used`는 `[true,false,false]`이므로

`if (used[i]) continue;` 조건을 회피하게 된다.

이후 `current`에 `nums[2]`를 삽입하고, `used[2]`를 `true`로 변경하면

`current`는 `[1,3]`이 된다.

그 다음 `backtrack()`을 재귀 호출하고,

`for loop`에서 `i=0~2`까지 진행하며 `[1,3,2]`를 만들고, `result`에 추가한다.

이후 **i=0** 일때의 모든 접근 경우의수를 찾았으니

`pop_back()`으로 `current` 전체를 비우고,

i=1 일 때의 작업을 반복 수행한다.

Solution update

```
class Solution {
public:
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> result;
        vector<int> current;
        unordered_map<int, int> counter;

        // count the frequency of each element
        for (int num : nums) {
```



```

        counter[num]++;
    }

    backtrack(result, current, counter, nums.size());
    return result;
}

private:
    void backtrack(vector<vector<int>>& result, vector<int>& current,
        unordered_map<int, int>& counter, int n) {
        if (current.size() == n) {
            result.push_back(current);
            return;
        }

        for (auto& [num, count] : counter) {
            if (count > 0) {
                current.push_back(num);
                counter[num]--;

                backtrack(result, current, counter, n);

                counter[num]++;
                current.pop_back();
            }
        }
    }
};

```

unordered_map을 이용한 풀이.

장점으로는 숫자가 중복되더라도 좀 더 빠르게 접근가능하다는 점이 장점이다.

[참조 링크](#)

위 문제는 unordered map 이용해서 풀어낸 방법.

47. Permutations II

layout: post title: 47. Permutations II category: leetcode date: 2025-01-21 13:59:00 +0900 description: <https://leetcode.com/problems/permutations-ii/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

47. Permutations II

Given a collection of numbers, `nums`, that might contain duplicates, return all possible unique permutations in any order.

Example 1:
Input: `nums = [1,1,2]`
Output:
`[[1,1,2],`
`[1,2,1],`
`[2,1,1]]`

Example 2:
Input: `nums = [1,2,3]`
Output: `[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]`

Solution

```
class Solution {
public:
    vector<vector<int>> results;
    set<vector<int>> myset;
    vector<int> current;
    vector<bool> used;
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        used.assign(nums.size(), false); // 초기화
        permutation(nums);
        for(auto element:myset){ // set은 index 접근 불가능하니까 auto로
            results.push_back(element);
        }
        return results;
    }
    void permutation(vector<int>& nums){
        if(nums.size() == current.size()){
            myset.insert(current); // set은 insert 쓰는 거 까먹지 말자
            return;
        }
        for(int i=0; i<nums.size(); i++){
            if(used[i]) continue;
            used[i] = true;
            current.push_back(nums[i]);
            permutation(nums);
            current.pop_back();
            used[i] = false; // 순간 까먹음. 다시 돌려줘야지
        }
    }
};
```

1번 문제 풀어봤으면 어렵지 않게 풀 수 있음.

근데 불필요한 set을 이용하기 때문에 개선하면 아래 코드가 된다

Solution2

```
class Solution {
public:
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        vector<vector<int>> results;
        vector<int> current;
        vector<bool> used(nums.size(), false);
        sort(nums.begin(), nums.end());
        // nums 배열을 정렬하여 중복된 숫자가 인접하도록
        backtrack(nums, current, used, results);
        return results;
    }

private:
    void backtrack(vector<int>& nums, vector<int>& current, vector<bool>& used,
vector<vector<int>>& results) {
        if (current.size() == nums.size()) {
            results.push_back(current);
            return;
        }
        for (int i = 0; i < nums.size(); i++) {
            // 중복된 숫자가 사용되는 것을 방지하기 위한 조건
            if (used[i] || (i > 0 && nums[i] == nums[i - 1] && !used[i - 1]))
                continue;
            used[i] = true;
            current.push_back(nums[i]);
            backtrack(nums, current, used, results);
            current.pop_back();
            used[i] = false;
        }
    }
};
```

121. Best Time to Buy and Sell Stock

layout: post title: 121. Best Time to Buy and Sell Stock category: leetcode
date: 2025-01-22 13:35:00 +0900 description:
<https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/>
envType=company&envId=google&favoriteSlug=google-thirty-days

img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

121. Best Time to Buy and Sell Stock

You are given an array `prices` where `prices[i]` is the price of a given stock on the *i*th day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Example 1:

Input: `prices = [7,1,5,3,6,4]`

Output: 5

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

Example 2:

Input: `prices = [7,6,4,3,1]`

Output: 0

Explanation: In this case, no transactions are done and the max profit = 0.

Solution (Failed)

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int index = 0;
        int current = 0;
        for(int i=0; i<prices.size(); i++)
        {
            for(int j=i; j<prices.size(); j++ )
            {
                if(prices[j]-prices[i]> current)
                {
                    current = prices[j]-prices[i];
                }
            }
        }
        return current;
    }
};
```

브루트 포스를 이용한 풀이.

예상했지만 당연히 시간초과를 하게된다.

Solution 1

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        if (prices.empty()) return 0;
        int minPrice = prices[0];
        int maxProfit = 0;

        for (int i = 1; i < prices.size(); i++) {
            minPrice = min(minPrice, prices[i]);
            maxProfit = max(maxProfit, prices[i] - minPrice);
        }
        return maxProfit;
    }
};
```

가장 최적의 방식.

Kadane's Algorithm 이라고 한다.

위 문제를 풀다보면 알게 되지만, Dynamic programming이다.

(하위 문제들의 결과를 저장하고, 이를 이용해서 전체 문제를 해결)

DP 문제를 해결할 때 쓰이는 알고리즘 중 하나이며,

위 문제를 점화식으로 표현하자면 아래와 같다. $dp[0] = 0$ \ 최소값 = $prices[0]$ \ $dp[i] = \max(dp[i-1], prices[i] - \text{최소값})$ \ 최소값 = $\min(\text{최소값}, prices[i])$

이를 고려해서 식을 만들어주면 된다. $O(n)$ 이므로 가장 효율적이다.

Solution 2

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        if (prices.size() < 2) return 0;
        return maxProfitHelper(prices, 0, prices.size() - 1);
    }

private:
    int maxProfitHelper(const vector<int>& prices, int left, int right) {
```

```

    if (left >= right) return 0;

    int mid = left + (right - left) / 2;

    int leftProfit = maxProfitHelper(prices, left, mid);
    int rightProfit = maxProfitHelper(prices, mid + 1, right);

    int minPrice = prices[left];
    for (int i = left + 1; i <= mid; ++i) {
        if (prices[i] < minPrice) {
            minPrice = prices[i];
        }
    }

    int maxPrice = prices[mid + 1];
    for (int i = mid + 2; i <= right; ++i) {
        if (prices[i] > maxPrice) {
            maxPrice = prices[i];
        }
    }

    int crossProfit = maxPrice - minPrice;

    return max({leftProfit, rightProfit, crossProfit});
}
};

```

Divide and conquer 이용하는 방식.

mergesort 할때처럼 중간을 잘라서 분할해주고, 두 개의 값 차이를 저장한뒤에 비교하면 된다.

$O(\log N)$

Solution3 (Solution 1 upgrade)

```

class Solution {
public:
    int maxProfit(std::vector<int>& prices) {
        int buy = prices[0];
        int profit = 0;
        for (int i = 1; i < prices.size(); i++) {
            if (prices[i] < buy) {
                buy = prices[i];
            } else if (prices[i] - buy > profit) {
                profit = prices[i] - buy;
            }
        }
        return profit;
    }
};

```

min max 호출을 없애서 더 빠르게만들

특이 사항 : std::vector와 같은 fully qualified name을 사용하면 컴파일러가 해당 타입이 표준 라이브러리에 있는 것을 빠르게 알 수 있어서, 약간의 시간향상이 있음.

136. Single Number

layout: post title: 136. Single Number category: leetcode date: 2025-01-22 06:02:00 +0900 description: <https://leetcode.com/problems/single-number/description/>

envType=company&envId=google&favoriteSlug=google-thirty-days
img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

136. Single Number

Given a non-empty array of integers `nums`, every element appears twice except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:
Input: `nums = [2,2,1]`
Output: `1`

Example 2:
Input: `nums = [4,1,2,1,2]`
Output: `4`

Example 3:
Input: `nums = [1]`
Output: `1`

Solution

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        unordered_map<int, int> frequency;
        for (int num : nums) {
```

```

        frequency[num]++;
    }
    for (int num : nums) {
        if (frequency[num] == 1) {
            return num;
        }
    }
    return 0;
}
};

```

내가풀려고 했던 풀이.

등장 횟수를 카운트하고, 1번만 등장했으면 return

Solution 2

```

class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int result = 0;
        for (int num : nums) {
            result ^= num;
        }
        return result;
    }
};

```

Bit 연산을 이용한 최적의 풀이.

\wedge 연산자는 XOR을 수행하는 연산자인데,

result에 num을 XOR한 결과를 계속해서 넣는다는 뜻이다.

즉, 같은 숫자가 등장한다면 XOR 연산으로 인해서 사라지게되므로,

짝이 없는 숫자만 남게된다.

1480. Running Sum of 1d Array

layout: post title: 1480. Running Sum of 1d Array category: leetcode date: 2025-01-22 10:21:00 +0900 description: <https://leetcode.com/problems/running-sum-of-1d-array/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

1480. Running Sum of 1d Array

Given an array `nums`. We define a running sum of an array as `runningSum[i] = sum(nums[0]...nums[i])`.

Return the running sum of `nums`.

Example 1:

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

Output: `[1,3,6,10]`

Explanation: Running sum is obtained as follows: `[1, 1+2, 1+2+3, 1+2+3+4]`.

Example 2:

Input: `nums = [1,1,1,1,1]`

Output: `[1,2,3,4,5]`

Explanation: Running sum is obtained as follows: `[1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1]`.

Example 3:

Input: `nums = [3,1,2,10,1]`

Output: `[3,4,6,16,17]`

Solution

```
class Solution {
public:
    int temp = 0;
    vector<int> runningSum(vector<int>& nums) {
        for(int i=0; i<nums.size(); i++){
            temp +=nums[i];
            nums[i] = temp;
        }
        return nums;
    }
};
```

169. Majority Element

layout: post title: 169. Majority Element category: leetcode date: 2025-01-22 16:32:00 +0900 description: <https://leetcode.com/problems/majority->

element/description/?
 envType=company&envId=google&favoriteSlug=google-thirty-days
 img: leetcode.png # Add image post (optional) fig-caption: # Add
 figcaption (optional)

169. Majority Element

Given an array `nums` of size `n`, return the majority element.

The majority element is the element that appears more than $\lfloor n / 2 \rfloor$ times. You may assume that the majority element always exists in the array.

Example 1:
 Input: `nums = [3,2,3]`
 Output: `3`

Example 2:
 Input: `nums = [2,2,1,1,1,2,2]`
 Output: `2`

Solution

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        unordered_map<int,int> um;
        pair<int,int> current = {0,0}; // 사용법 숙지
        for(auto num:nums){
            um[num]++;
        }
        for(auto element:um){
            if(element.second > current.second) current = element; //사용법 숙지
        }
        return current.first;
    }
};
```

pair랑 map의 원소 꺼내올때 쓰는 법 순간 잊고 있었음

시간복잡도 $O(n)$ 공간복잡도 $O(n)$

Solution 2 (Advanced)

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int count = 0;
        int candidate = 0;

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

        // Phase 2: 확인
        count = 0;
        for(int num : nums) {
            if(num == candidate) {
                count++;
            }
        }

        if(count > nums.size() / 2) {
            return candidate;
        } else {
            // 문제의 조건에 따르면 항상 majority element가 있다고 가정하므로 여기까지
            // 도달하지 않습니다.
            return -1; // 예외 처리
        }
    }
};
```

Boyer-Moore Majority Vote 알고리즘

<https://sgc109.github.io/2020/11/30/boyer-moore-majority-vote-algorithm/>

시간있을 때 한 번 보자...

191. Number of 1 Bits

layout: post title: 191. Number of 1 Bits category: leetcode date: 2025-01-22 20:12:00 +0900 description: <https://leetcode.com/problems/number-of-1-bits/description/>

envType=company&envId=google&favoriteSlug=google-thirty-days
img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

191. Number of 1 Bits

Given a positive integer n , write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

Example 1:
Input: $n = 11$
Output: 3
Explanation:
The input binary string 1011 has a total of three set bits.

Example 2:
Input: $n = 128$
Output: 1
Explanation:
The input binary string 10000000 has a total of one set bit.

Example 3:
Input: $n = 2147483645$
Output: 30
Explanation:
The input binary string 11111111111111111111111111101 has a total of thirty set bits.

Solution

```
class Solution {
public:
    int cnt = 0;
    int hammingWeight(int n) {
        int temp = calculation(n);
        return temp+cnt;
    }
    int calculation(int n){
        if(n==1) return 1;
        int i = 0;
        while(pow(2,i)<n){
            i++;
        }
        cnt++;
        if (n-pow(2,i)==0) return 0;
        return calculation(n-pow(2,i-1));
    }
};
```

내 풀이지만 시간복잡도가 $O(\log n)$ 이라서 아직 부족함.

비트연산을 이용하면 더 빨라질 것

Solution 2 (Advanced)

```
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1); // Check if the least significant bit is 1
            n >>= 1;          // Right shift n by 1 bit
        }
        return count;
    }
};
```

개선된 코드. bit shift 이용해서 해결. 시간복잡도 $O(1)$ 공간복잡도 $O(1)$ 비트 비교를 이용해서 접근하는 방법

```
count += (n & 1);
```

이 코드가 핵심인데, $n \& 1$ 은 n 과 1의 and 연산이고, $O(1)$ 시간복잡도.

즉, 마지막 자리수에 1이 있는지 확인하는 것.

이후 bit shift 를 한 뒤에 다시 1이 있는지를 확인하는 반복작업.

다만, 입력이 고정되지 않은 가변 크기 비트 연산 환경이라면 반복문이 입력 크기에 따라 달라져 $O(\log(n))$ 이 될 수 있음.

2094. Finding 3-Digit Even Numbers

layout: post title: 2094. Finding 3-Digit Even Numbers category: leetcode
date: 2025-01-22 15:09:00 +0900 description:
<https://leetcode.com/problems/finding-3-digit-even-numbers/description/>
envType=company&envId=google&favoriteSlug=google-thirty-days
img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

2094. Finding 3-Digit Even Numbers

You are given an integer array `digits`, where each element is a digit. The array may contain duplicates.

You need to find all the unique integers that follow the given requirements:

The integer consists of the concatenation of three elements from `digits` in any arbitrary order. The integer does not have leading zeros. The integer is even. For example, if the given `digits` were `[1, 2, 3]`, integers `132` and `312` follow the requirements.

Return a sorted array of the unique integers.

Example 1:

Input: `digits = [2,1,3,0]`

Output: `[102,120,130,132,210,230,302,310,312,320]`

Explanation: All the possible integers that follow the requirements are in the output array.

Notice that there are no odd integers or integers with leading zeros.

Example 2:

Input: `digits = [2,2,8,8,2]`

Output: `[222,228,282,288,822,828,882]`

Explanation: The same digit can be used as many times as it appears in `digits`.

In this example, the digit `8` is used twice each time in `288`, `828`, and `882`.

Example 3:

Input: `digits = [3,7,5]`

Output: `[]`

Explanation: No even integers can be formed using the given `digits`.

Solution (Failed)

```
class Solution {
public:
    vector<bool> visited;
    set<int> temporal_set;
    vector<int> current;
    vector<int> findEvenNumbers(vector<int>& digits) {
        visited.assign(digits.size(), false);
        permutation(digits);
        vector<int> results(temporal_set.begin(), temporal_set.end());
        sort(results.begin(), results.end());
        return results;
    }
    void permutation(vector<int>& digits){
        if(current.size()==3){
```

```

        if(current[0] == 0) return;
        if(current[2]%2 ==1) return;
        else{
            std::ostringstream oss;
            for (const auto &num : current) {
                oss << num;
            }
            std::string result = oss.str();
            int number = stoi(result);
            temporal_set.insert(number);
            return;
        }
    }
    for(int i=0; i<digits.size(); i++){
        if(visited[i]==true) continue;
        visited[i] = true;
        current.push_back(digits[i]);
        permutation(digits);
        visited[i] = false;
        current.pop_back();
    }
}
};

```

텍걸이로 가끔 pass 하는 잘못된 풀이

Time limit에 걸려서 틀렸고, 아래에서 순차적으로 개선 프로세스를 공개함

```

if(current.size()==3){
    if(current[0] == 0) return;
    if(current[2]%2 ==1) return;
    else{
        int number = current[0] * 100 + current[1] * 10 + current[2];
        if(current[0] != 0 && current[2] % 2 == 0) {
            temporal_set.insert(number);
        }
    }
    return;
}

```

우선, else 부분만 바꿨음.

기존에 숫자->문자->문자열->숫자로 변환하는 비효율적인 과정 제거,

그냥 숫자만 있어도 어차피 3자리 숫자기때문에 x100, x10, 해주면 그만임

그 다음은 set-> unordered_set

hash 이용해서 속도 약간 빠르게 함. (170ms 줄어듦)

```

unordered_set<int> temporal_set;

```

추가로 한 번 더

```
unordered_map<int, int> countMap;
```

이렇게 고쳐주자.

이렇게 하면 visited를 쓰지않고 카운트를 이용해서 풀 수 있다.

visited vector를 사용하게되면, 배열을 생성해야하는 인풋이 커질수록 공간을 소모하고 체크하는 과정에서 연산이 너무 커진다.

```
class Solution {
public:
    unordered_set<int> temporal_set;
    vector<int> current;

    vector<int> findEvenNumbers(vector<int>& digits) {
        unordered_map<int, int> countMap;
        for (int digit : digits) {
            countMap[digit]++; //숫자에 대응하는 value값을 미리 증가시켜놓기
        }
        permutation(countMap, 0);
        vector<int> results(temporal_set.begin(), temporal_set.end());
        sort(results.begin(), results.end());
        return results;
    }

    void permutation(unordered_map<int, int>& countMap, int index) {
        if (current.size() == 3) {
            if (current[0] == 0) return;
            if (current[2] % 2 == 1) return;

            int number = current[0] * 100 + current[1] * 10 + current[2];
            temporal_set.insert(number);
            return;
        }

        for (auto& [digit, count] : countMap) {
            if (count == 0) continue; //만약 count를 다 쓰면 숫자 없는거

            count--; //visited = true 대신 count
            current.push_back(digit);
            permutation(countMap, index + 1);
            current.pop_back();
            count++;
        }
    }
};
```


이렇게 하면 $O(3!)$ (자릿수 3)

27. Remove Element

layout: post title: 27. Remove Element category: leetcode date: 2025-01-22 07:18:00 +0900 description: <https://leetcode.com/problems/remove-element/description/>
envType=company&envId=google&favoriteSlug=google-six-months
img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

27. Remove Element

Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` in-place. The order of the elements may be changed. Then return the number of elements in `nums` which are not equal to `val`.

Consider the number of elements in `nums` which are not equal to `val` be `k`, to get accepted, you need to do the following things:

Change the array `nums` such that the first `k` elements of `nums` contain the elements which are not equal to `val`. The remaining elements of `nums` are not important as well as the size of `nums`. Return `k`. Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int val = ...; // Value to remove
int[] expectedNums = [...]; // The expected answer with correct length. // It is sorted with no values equaling val.
```

```
int k = removeElement(nums, val); // Calls your implementation
```

```
assert k == expectedNums.length;
sort(nums, 0, k); // Sort the first k elements of nums
for (int i = 0; i < actualLength; i++) {
    assert nums[i] == expectedNums[i];
}
// If all assertions pass, then your solution will be accepted.
```

Example 1:

Input: `nums = [3,2,2,3]`, `val = 3`

Output: `2`, `nums = [2,2,_,_]`

Explanation: Your function should return `k = 2`, with the first two elements of `nums` being `2`.

It does not matter what you leave beyond the returned `k` (hence they are underscores).

Example 2:

Input: `nums = [0,1,2,2,3,0,4,2]`, `val = 2`

Output: `5`, `nums = [0,1,4,0,3,_,_,_]`

Explanation: Your function should return `k = 5`, with the first five elements of `nums` containing `0`, `0`, `1`, `3`, and `4`.

Note that the five elements can be returned in any order.
It does not matter what you leave beyond the returned k (hence they are underscores).

멍청하게 풀어서 반례를 못잡아서 이것도 틀림...

아래 접근방법처럼 깔끔하게 접근했어야했음.

나는 Swap 하는 방식으로 접근했는데, 불필요한 시간낭비임.

어차피 순서는 상관없기때문에 계속해서 갱신하는 방식으로 진행하는게 맞음.

Solution

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int index = 0; // 새로운 배열의 유효 인덱스
        for (int i = 0; i < nums.size(); ++i) {
            if (nums[i] != val) {
                nums[index] = nums[i];
                ++index;
            }
        }
        return index;
    }
};
```

58. Length of Last Word

layout: post title: 58. Length of Last Word category: leetcode date: 2025-01-22 11:00:00 +0900 description:
<https://leetcode.com/problems/length-of-last-word/description/> img:
leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

58. Length of Last Word

Given a string *s* consisting of words and spaces, return the length of the last word in the string.

A word is a maximal substring consisting of non-space characters only.

Example 1:
Input: *s* = "Hello World"

Output: 5

Explanation: The last word is "World" with length 5.

Example 2:

Input: s = " fly me to the moon "

Output: 4

Explanation: The last word is "moon" with length 4.

Example 3:

Input: s = "luffy is still joyboy"

Output: 6

Explanation: The last word is "joyboy" with length 6.

Solution

```
class Solution {
public:
    int count = 0;
    int lengthOfLastWord(string s) {
        for(int i=s.size()-1; i>=0; i--){
            if(s[i]!=' '){
                count++;
            }
            else if (s[i]==' ' && count >0) return count;
        }
        return count;
    }
};
```

```
for(int i=s.size()-1; i>=0; i--)
```

이 부분에서,

```
for(int i=s.size()-1; i>0; i--)
```

처럼 만들었는데, 인덱스 계산 잘못해서 틀렸었음

628. Maximum Product of Three Numbers

layout: post title: 628. Maximum Product of Three Numbers category: leetcode date: 2025-01-22 11:18:00 +0900 description: <https://leetcode.com/problems/maximum-product-of-three-numbers/description/> envType=company&envId=google&favoriteSlug=google-thirty-days img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

628. Maximum Product of Three Numbers

Given an integer array `nums`, find three numbers whose product is maximum and return the maximum product.

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

Example 2:
Input: `nums = [1,2,3,4]`
Output: 24

Example 3:
Input: `nums = [-1,-2,-3]`
Output: -6

쉬운문제인데 생각하기 귀찮아서 사실 답지봄

Solution

```
class Solution {
public:
    int maximumProduct(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int n = nums.size();
        return max(nums[n-1]*nums[n-2]*nums[n-3], nums[n-1]*nums[0]*nums[1]);
    }
};
```

특이 사항 : 최대 수를 얻으려면 음수 2개 * 양수 제일큰거 하나인 예가 있음.

그냥 max 이용해서 일반해랑 특수해 둘 중에 큰거 쓰면됨

특이사항 2 : int n = nums.size(); 선언을 안쓰고

n을 전부 nums.size()로 바꿔서 코드 줄을 줄였는데, 실제로는 함수호출시간이 더 걸리므로 느려짐

88. Merge Sorted Array

layout: post title: 88. Merge Sorted Array category: leetcode date: 2025-01-22 06:02:00 +0900 description:

<https://leetcode.com/problems/merge-sorted-array/description/?envType=company&envId=google&favoriteSlug=google-six-months>

img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

88. Merge Sorted Array

You are given two integer arrays `nums1` and `nums2`, sorted in non-decreasing order, and two integers `m` and `n`, representing the number of elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array `nums1`. To accommodate this, `nums1` has a length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to 0 and should be ignored. `nums2` has a length of `n`.

Example 1:

Input: `nums1 = [1,2,3,0,0,0]`, `m = 3`, `nums2 = [2,5,6]`, `n = 3`

Output: `[1,2,2,3,5,6]`

Explanation: The arrays we are merging are `[1,2,3]` and `[2,5,6]`.

The result of the merge is `[1,2,2,3,5,6]` with the underlined elements coming from `nums1`.

Example 2:

Input: `nums1 = [1]`, `m = 1`, `nums2 = []`, `n = 0`

Output: `[1]`

Explanation: The arrays we are merging are `[1]` and `[]`.

The result of the merge is `[1]`.

Example 3:

Input: `nums1 = [0]`, `m = 0`, `nums2 = [1]`, `n = 1`

Output: `[1]`

Explanation: The arrays we are merging are `[]` and `[1]`.

The result of the merge is `[1]`.

Note that because $m = 0$, there are no elements in `nums1`. The `0` is only there to ensure the merge result can fit in `nums1`.

제대로 인덱스 생각안하고 풀어서 30분이나걸림

문제 풀기전에 문제를 정확히 보고 풀자

Solution

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        if(m==0) nums1 = nums2;
        if(n==0) return;
        for(int i=0; i<n; i++){
            nums1[i+m] = nums2[i];
        }
        sort(nums1.begin(),nums1.end());
    }
};
```

1768. Merge Strings Alternately

layout: post title: 1768. Merge Strings Alternately category: leetcode date: 2025-01-24 21:42:00 +0900 description: <https://leetcode.com/problems/merge-strings-alternately/description/?envType=company&envId=google&favoriteSlug=google-thirty-days> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

1768. Merge Strings Alternately

You are given two strings `word1` and `word2`. Merge the strings by adding letters in alternating order, starting with `word1`. If a string is longer than the other, append the additional letters onto the end of the merged string.

Return the merged string.

```
Example 1:
Input: word1 = "abc", word2 = "pqr"
Output: "apbqcr"
Explanation: The merged string will be merged as so:
word1:  a   b   c
word2:  p   q   r
```

```
word2:   p   q   r
merged: a p b q c r
```

Example 2:

Input: word1 = "ab", word2 = "pqrs"

Output: "apbqrs"

Explanation: Notice that as word2 is longer, "rs" is appended to the end.

```
word1:  a   b
```

```
word2:   p   q   r   s
```

```
merged: a p b q   r
```

Example 3:

Input: word1 = "abcd", word2 = "pq"

Output: "apbqcd"

Explanation: Notice that as word1 is longer, "cd" is appended to the end.

```
word1:  a   b   c   d
```

```
word2:   p   q
```

```
merged: a p b q c   d
```

Solution

```
class Solution {
public:
    string results;
    bool flag;
    string mergeAlternately(string word1, string word2) {
        mergeString(word1, word2);
        return results;
    }
    void mergeString(string word1, string word2){
        if(word1.size() > word2.size()) flag = true;
        int minSize = (word1.size() < word2.size()) ? word1.size() : word2.size();
        int maxSize = (word1.size() > word2.size()) ? word1.size() : word2.size();
        for(int i=0; i<minSize; i++){
            results += word1[i];
            results += word2[i];
        }
        if(flag){
            for(int i=minSize; i<maxSize; i++){
                results += word1[i];
            }
        }
        else{
            for(int i=minSize; i<maxSize; i++){
                results += word2[i];
            }
        }
    }
};
```

```

    }
}

};

```

풀리긴 하지만 쓸모없는 과정이 몇 개 있다.

우선, member var를 만들어서 이용하는 것 보다, 함수내에서 선언하여 parameter로 전달하는 것이 더 빠름.

그리고,

```

if(word1.size() > word2.size()) flag = true;
int minSize = (word1.size() < word2.size()) ? word1.size() : word2.size();
int maxSize = (word1.size() > word2.size()) ? word1.size() : word2.size();

```

여기서 이렇게 체크를 해주는 것이 불필요함.

Solution 2

```

class Solution {
public:
    string mergeAlternately(string word1, string word2) {
        int i = 0, j = 0;
        string result;
        while (i < word1.size() && j < word2.size()) {
            result += word1[i];
            result += word2[j];
            i++;
            j++;
        }
        while (i < word1.size()) {
            result += word1[i];
            i++;
        }
        while (j < word2.size()) {
            result += word2[j];
            j++;
        }
        return result;
    }
};

```

여기서는


```
while (i < word1.size() && j < word2.size())
```

이런 식으로 조건을 걸어주는데, 이렇게 하면 둘이 겹치는 길이까지만 동작가능.

28. Find the Index of the First Occurrence in a String

layout: post title: 28. Find the Index of the First Occurrence in a String
category: leetcode date: 2025-01-24 22:27:00 +0900 description:
<https://leetcode.com/problems/find-the-index-of-the-first-occurrence-in-a-string/description/>
envType=company&envId=google&favoriteSlug=google-thirty-days
img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

28. Find the Index of the First Occurrence in a String

Given two strings *needle* and *haystack*, return the index of the first occurrence of *needle* in *haystack*, or -1 if *needle* is not part of *haystack*.

Example 1:
Input: haystack = "sadbutsad", needle = "sad"
Output: 0
Explanation: "sad" occurs at index 0 and 6.
The first occurrence is at index 0, so we return 0.

Example 2:
Input: haystack = "leetcode", needle = "leeto"
Output: -1
Explanation: "leeto" did not occur in "leetcode", so we return -1.

Solution

```
class Solution {  
public:  
    int strStr(string haystack, string needle) {  
        int hs = haystack.size();  
        int ns = needle.size();  
        for(int i = 0; i < hs - ns + 1; i++){  
            if(haystack[i] == needle[0]){  
                if (ns == 1) return i;  
            }  
        }  
        return -1;  
    }  
};
```

```

        for(int j=1; j<ns; j++){
            if(haystack[i+j] == needle[j]) {
                if(j==ns-1) return i;
                continue;
            }
            else break;
        }
    }
    return -1;
}
};

```

find() 사용하지 않고 풀기.

test case 3개에서 막혔는데, 간과한 부분

1. needle size가 1인 경우 if (ns == 1) return i; 를 추가안해서 틀렸음
2. 1을 잘못 이해하고, 생각없이 needle과 haystack size가 같은 경우 return 0; 했다가 틀림

189. Rotate Array

layout: post title: 189. Rotate Array category: leetcode date: 2025-01-25 15:02:00 +0900 description: <https://leetcode.com/problems/rotate-array/description/>
 envType=company&envId=google&favoriteSlug=google-thirty-days
 img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

189. Rotate Array

Given an integer array `nums`, rotate the array to the right by `k` steps, where `k` is non-negative.

Example 1:
 Input: `nums = [1,2,3,4,5,6,7]`, `k = 3`
 Output: `[5,6,7,1,2,3,4]`
 Explanation:
 rotate 1 steps to the right: `[7,1,2,3,4,5,6]`
 rotate 2 steps to the right: `[6,7,1,2,3,4,5]`
 rotate 3 steps to the right: `[5,6,7,1,2,3,4]`

Example 2:
 Input: `nums = [-1,-100,3,99]`, `k = 2`
 Output: `[3,99,-1,-100]`
 Explanation:

```
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]
```

Solution

```
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        deque<int> dq;
        for(int i=0; i<nums.size(); i++){
            dq.push_back(nums[i]);
        }
        for(int i=0; i<k; i++){
            int temp = dq.back();
            dq.pop_back();
            dq.push_front(temp);
        }
        nums.assign(dq.begin(), dq.end()); //iterator 지원해주니까 이렇게 이용하면됨
    }
};
```

deque(double-ended queue) 사용해서 풀음

시간복잡도 $O(N)$ 공간복잡도 $O(N)$

만약 공간복잡도를 줄이고싶다면 아래 풀이

```
void rotate(std::vector<int>& nums, int k) {
    int n = nums.size();
    k = k % n; // k가 배열 크기보다 클 수 있으므로 나머지 연산
    if (k == 0) return;
    // Step 1: Reverse the entire array
    std::reverse(nums.begin(), nums.end());
    // Step 2: Reverse the first k elements
    std::reverse(nums.begin(), nums.begin() + k);
    // Step 3: Reverse the rest of the array
    std::reverse(nums.begin() + k, nums.end());
}
```

공간복잡도는 $O(1)$ 이된다.

30. Substring with Concatenation of All Words

layout: post title: 30. Substring with Concatenation of All Words category: leetcode date: 2025-01-26 01:33:00 +0900 description: <https://leetcode.com/problems/substring-with-concatenation-of-all-words/description/?envType=company&envId=google&favoriteSlug=google-thirty-days> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

30. Substring with Concatenation of All Words

You are given a string *s* and an array of strings *words*. All the strings of *words* are of the same length.

A concatenated string is a string that exactly contains all the strings of any permutation of *words* concatenated.

For example, if *words* = ["ab","cd","ef"], then "abcdef", "abefcd", "cdabef", "cdefab", "efabcd", and "efcdab" are all concatenated strings. "acdbef" is not a concatenated string because it is not the concatenation of any permutation of *words*. Return an array of the starting indices of all the concatenated substrings in *s*. You can return the answer in any order.

Example 1:

Input: *s* = "barfoothefoobarman", *words* = ["foo","bar"]

Output: [0,9]

Explanation:

The substring starting at 0 is "barfoo". It is the concatenation of ["bar","foo"] which is a permutation of *words*.

The substring starting at 9 is "foobar". It is the concatenation of ["foo","bar"] which is a permutation of *words*.

Example 2:

Input: *s* = "wordgoodgoodgoodbestword", *words* = ["word","good","best","word"]

Output: []

Explanation:

There is no concatenated substring.

Example 3:

Input: *s* = "barfoofoobarthefoobarman", *words* = ["bar","foo","the"]

Output: [6,9,12]

Explanation:

The substring starting at 6 is "foobarthe". It is the concatenation of ["foo","bar","the"].

The substring starting at 9 is "barthefoo". It is the concatenation of ["bar","the","foo"].

The substring starting at 12 is "thefoobar". It is the concatenation of ["the","foo","bar"].

Solution (Failed)

```
class Solution {
public:
    vector<string> perms;

    void permutation(vector<string>& words, vector<string>& current , vector<bool>
visited){
        if(words.size()== current.size()){
            string temp;
            for(int i=0; i<current.size();i++){

                temp += current[i];
            }
            perms.push_back(temp);
            return;
        }
        for(int i=0; i<words.size(); i++){
            if(visited[i]) continue;
            visited[i] = true;
            current.push_back(words[i]);
            permutation(words,current,visited);
            current.pop_back();
            visited[i] = false;
        }
    }

    vector<int> checkContain(string s, vector<string> perms){
        unordered_set<int> mySet;

        for(int i=0; i<perms.size(); i++){
            int index=0;
            while(index<=s.size()){
                index = s.find(perms[i],index);
                if(index ==-1) continue;
                mySet.insert(index);
                index++;
            }
        }
        vector<int> results(mySet.begin(), mySet.end());
        return results;
    }

    vector<int> findSubstring(string s, vector<string>& words) {
        // s안에 들어있는 words의 permutation을 찾아내고, index를 각각 집어넣는 함수
        if(words.size()==0){
            return {0};
        }
        vector<string> current;
        vector<bool> visited;
```

```

        visited.assign(words.size(), false);
        permutation(words, current, visited);
        vector<int> kyj = checkContain(s, perms);
        return kyj;
    }
};

```

내 접근 방식 : 순열을 직접 만들어서, 만들어진 순열과 비교 후 find하여 인덱스를 얻는다.

Test Case 틀렸던 이유

1. 문자열을 찾지 못해서 -1을 return 하는 경우에 대한 handling X
2. find를 한 번 만 수행해서, 같은 문자열이 또 나오는 경우에 대한 handling X

최종적으로 못 푼 이유 (Time Limit Exceeded)

```

s = "ffffffffffffffffffffffffffffffffffff"
words =
["a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a"]

```

위 케이스에서 시간초과로 실패함.

시간초과의 핵심적 이유

1. 순열 생성 방식의 비효율성
 - words의 크기가 k라고 하면, 순열의 개수는 k!이므로, $O(k!)$
2. 중복된 인덱스 처리
 - s.find()를 수행하는데, find는 $O(n)$ 이고, k!에 대해서 수행하므로 $O(k! * n)$ 이 된다.
3. 조건활용도 부족
 - 문제에서 words는 같은크기들이라고 말했는데, 이는 Sliding window문제라는 뜻이다.

Solution

```

class Solution {
public:
    vector<int> findSubstring(string s, vector<string>& words) {
        if (s.empty() || words.empty()) return {};

        // 단어 길이와 총 길이 계산
        int wordLen = words[0].size();
        int wordCount = words.size();
        int totalLen = wordLen * wordCount;

        // 단어 빈도 저장
        unordered_map<string, int> wordFreq;
        for (const auto& word : words) {
            wordFreq[word]++;
        }
    }
};

```

```

    }

    vector<int> result;

    // 슬라이딩 윈도우 탐색
    for (int i = 0; i < wordLen; ++i) {
        int left = i, count = 0;
        unordered_map<string, int> windowFreq;

        for (int right = i; right + wordLen <= s.size(); right += wordLen) {
            string word = s.substr(right, wordLen);

            if (wordFreq.find(word) != wordFreq.end()) {
                windowFreq[word]++;
                count++;

                // 단어 빈도가 초과되면 왼쪽 포인터 이동
                while (windowFreq[word] > wordFreq[word]) {
                    string leftWord = s.substr(left, wordLen);
                    windowFreq[leftWord]--;
                    count--;
                    left += wordLen;
                }

                // 모든 단어가 포함되면 시작 인덱스 저장
                if (count == wordCount) {
                    result.push_back(left);
                }
            } else {
                // 유효하지 않은 단어 발견 시 초기화
                windowFreq.clear();
                count = 0;
                left = right + wordLen;
            }
        }
    }

    return result;
}
};

```

핵심 개념은 아래와 같다.

words의 모든 word들을 이용한 연속된 string을 찾으면된다.

어차피 words에서 word들의 길이가 고정되어있으므로,

s 문자열을 word size만큼 분할해서 체크하는 것이다.

그리고, map을 이용해서 각각을 count 해주고,

count의 총합을 이용하면 word들을 모두 사용했는지 확인이 가능하다.

예를들어,

```
s = "barfoothefoobarman"
words = ["foo", "bar"]
```

라고 할때,

bar에는 word가 있으므로, {"foo" : 0, "bar": 1}이되고,

foo에도 word가 있으므로 {"foo" : 1, "bar": 1}

이후 the에서는 없으므로 결과 return후 {"foo" : 0, "bar": 0}으로 초기화.

다시 foo 가 있으므로 {"foo" : 1, "bar": 0}

이런식으로 반복하면 된다.

이 코드는 윈도우 탐색에 $O(n)$, 단어 비교에는 $O(k)$ (k 는 상수라서 빼도됨) 공간복잡도는 wordFreq와 windowFreq는 각각 $O(k)$, result 는 $O(n)$ 즉, $O(k+n)$

494. Target Sum

layout: post title: 494. Target Sum category: leetcode date: 2025-01-26 03:20:00 +0900 description: <https://leetcode.com/problems/target-sum/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

494. Target Sum

You are given an integer array `nums` and an integer `target`.

You want to build an expression out of `nums` by adding one of the symbols '+' and '-' before each integer in `nums` and then concatenate all the integers.

For example, if `nums` = [2, 1], you can add a '+' before 2 and a '-' before 1 and concatenate them to build the expression "+2-1". Return the number of different expressions that you can build, which evaluates to `target`.

Example 1:

Input: `nums` = [1,1,1,1,1], `target` = 3

Output: 5

Explanation: There are 5 ways to assign symbols to make the sum of `nums` be `target` 3.

-1 + 1 + 1 + 1 + 1 = 3

+1 - 1 + 1 + 1 + 1 = 3

+1 + 1 - 1 + 1 + 1 = 3

+1 + 1 + 1 - 1 + 1 = 3

+1 + 1 + 1 + 1 - 1 = 3

Example 2:
Input: nums = [1], target = 1
Output: 1

Solution 1 (DFS)

```
class Solution {
public:
    void DFS(vector<int>& nums,int target,int curr,int index,int& result){
        if(index == nums.size()){
            if(curr == target)
                ++result;
            return ;
        }
        DFS(nums,target,curr - nums[index],index+1,result);
        DFS(nums,target,curr + nums[index],index+1,result);
    }

    int findTargetSumWays(vector<int>& nums, int target) {
        int result = 0 ;
        DFS(nums,target,0,0,result);
        return result;
    }
};
```

DFS에서 자주 나오는 아주 중요한 문제.

'+', '-' 라는 Binary를 생각해보면,

nums의 숫자가 k 개일때 총 가능한 경우의 수는 2^k 가지임.

그 중에서 결과를 만족하는 경우를 찾는 것이므로 DFS로 풀 수있음.

'-' 이동 또는 '+' 이동 둘 중 하나이므로, Binary Tree에서의 이동과 같다.

그러므로, DFS left, right 하듯이 접근해주는 것이 관건.

시간 복잡도 : $O(2^N)$ 공간 복잡도 : $O(N)$

Solution2 (Advanced)

```
class Solution {
public:
    void DFS(vector<int>& nums, int target, int curr, int index, int& result, int
```

```

    remaining_sum) {
        // Pruning: 현재 합으로 target에 도달할 수 없으면 탐색 중단
        if (abs(curr - target) > remaining_sum)
            return;

        if (index == nums.size()) {
            if (curr == target)
                ++result;
            return;
        }

        DFS(nums, target, curr - nums[index], index + 1, result, remaining_sum -
nums[index]);
        DFS(nums, target, curr + nums[index], index + 1, result, remaining_sum +
nums[index]);
    }

    int findTargetSumWays(vector<int>& nums, int target) {
        int result = 0;
        int remaining_sum = 0;
        for (int num : nums)
            remaining_sum += num;
        DFS(nums, target, 0, 0, result, remaining_sum);
        return result;
    }
};

```

하지만 1번 풀이의 한계점은, DFS로만 구성되었고 backtracking 하지 않는다는 것.

Backtracking은 Pruning(가지치기) 작업이 있어야만 한다.

Permutation을 만들 때, visited를 이용하는 것도 그 이유 중 하나.

그래서

Solution 3 (DP)

```

int findTargetSumWays(vector<int>& nums, int target) {
    int totalSum = accumulate(nums.begin(), nums.end(), 0);
    // 간단하게 누적 합 구하는 함수 accumulate
    // 3번째 파라미터는 초기 시작 값임.

    if ((totalSum - target) < 0 || (totalSum - target) % 2 != 0) {
        return 0;
    }

    int subsetSum = (totalSum - target) / 2;

    // DP vector to store the number of ways to achieve each sum

```

```

vector<int> dp(subsetSum + 1, 0);
// dp에는 subsetSum 까지의 모든 정보들을 저장해야하므로 +1 까지 저장
dp[0] = 1; // Base case: one way to achieve sum 0 (empty subset)

// Update DP array
for (int num : nums) {
    for (int j = subsetSum; j >= num; --j) {
        dp[j] += dp[j - num];
    }
}

return dp[subsetSum];
}

```

첫 번째 풀이에서의 문제점은, 모든 경우의 수 32가지를 다 돌려본다는 점임.

공통분모가 존재하는 경우에 예외처리를 통해서 불필요한 계산은 생략해야함.

→ Memoization 을 통한 해결

dp라는 벡터를 만들어서, **부분합** 들을 기록해 놓으면 됨.

예를들어,

```

vector<int> nums = {1, 1, 2, 2, 2, 3, 3, 6};
int target = 6;

```

위와 같은 경우,

dp에는 최종적으로 이렇게 기록되어야함

[1,2,4,8,10,14,18,18]

target이 6이므로, 총 7개의 공간이 필요하고, (합이 0인경우는 1로 고정)

dp[6] = 18 은, sum이 6인 경우는 모두 18가지가 있다는 얘기.

```

for (int num : nums) {
    for (int j = subsetSum; j >= num; --j) {
        dp[j] += dp[j - num];
    }
}

```

이 부분이 DP를 업데이트 하는 부분이다.

뒷자리부터 채우면서 진행하고, 순서는 아래와 같다.

첫 번째 num 인 1에서

```
dp[7] += dp[6]
dp[6] += dp[5]
...
dp[1] += dp[0]
```

현재 dp의 나머지는 전부 0으로 초기화 되어있으므로,

dp 는 [1,0,0,0,0,0,0] 이 된다.

그 다음 num인 1에서도 똑같은 점화식이 진행됨.

```
dp[7] += dp[6]
dp[6] += dp[5]
...
dp[1] += dp[0]
```

dp는 [1,2,1,0,0,0,0] 이 된다.

이런 식으로 반복해주면, 최종 dp를 얻을 수 있다.

layout: post title: category: leetcode date: 2025-01-27 12:27:00 +0900
description: img: leetcode.png # Add image post (optional) fig-caption: #
Add figcaption (optional)

17. Letter Combinations of a Phone Number

layout: post title: 17. Letter Combinations of a Phone Number category:
leetcode date: 2025-01-27 12:58:00 +0900 description:
<https://leetcode.com/problems/letter-combinations-of-a-phone-number/description/>
envType=company&envId=google&favoriteSlug=google-thirty-days
img: leetcode.png # Add image post (optional) fig-caption: # Add
figcaption (optional)

17. Letter Combinations of a Phone Number

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

Example 2:
Input: digits = ""
Output: []

Example 3:
Input: digits = "2"
Output: ["a","b","c"]

Solution (Failed)

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        if (digits.empty()) return {}; // 입력이 비어있을 경우 빈 결과 반환

        vector<string> results;
        string current = "";
        vector<bool> visited;
        visited.assign(digits.size(), false);
        unordered_map<char, string> keypad = {
            {'2', "abc"}, {'3', "def"}, {'4', "ghi"}, {'5', "jkl"},
            {'6', "mno"}, {'7', "pqrs"}, {'8', "tuv"}, {'9', "wxyz"}
        };
        getperm(digits, keypad, results, visited, current);
        return results;
    }

    void getperm(string digits, unordered_map<char, string> keypad,
vector<string>& results, vector<bool>& visited, string& current) {
        if (digits.size() == current.size()) {
            results.push_back(current);
            return;
        }
        for (int i = 0; i < digits.size(); i++) {
            if (visited[i]) continue;
            visited[i] = true;

            // 모든 가능한 문자를 하나씩 추가
            for (char c : keypad[digits[i]]) {
                current.push_back(c); // 문자 추가
            }
        }
    }
};
```

```

        getperm(digits, keypad, results, visited, current);
        current.pop_back(); // 복구
    }

    visited[i] = false; // 방문 상태 복구
}
}
};

```

순열을 이용해서 접근하려고 했음.

그런데, 순서가 상관없다고 하는 부분이 있어서, 틀려버림.

결국은 조합으로 접근해야 했었음

Solution (Combination)

```

class Solution {
public:
    vector<string> letterCombinations(string digits) {
        if (digits.empty()) return {}; // 입력이 비어있을 경우 빈 결과 반환

        vector<string> results;
        string current = "";
        unordered_map<char, string> keypad = {
            {'2', "abc"}, {'3', "def"}, {'4', "ghi"}, {'5', "jkl"},
            {'6', "mno"}, {'7', "pqrs"}, {'8', "tuv"}, {'9', "wxyz"}
        };
        combine(digits, keypad, results, current, 0);
        return results;
    }

    void combine(string& digits, unordered_map<char, string>& keypad,
        vector<string>& results, string& current, int index) {
        if (index == digits.size()) {
            results.push_back(current);
            return;
        }
        // 조합으로 풀거라 visited 필요가없음
        for (char c : keypad[digits[index]]) {
            current.push_back(c); // 현재 자리 문자 추가
            combine(digits, keypad, results, current, index + 1); // 다음 자리 처리
            current.pop_back(); // 복구
        }
    }
};

```

순열에 비해서 조합은 쉽다

visited 관련된 부분 다 제거해주면 그만임

시간 복잡도 : $O(3^N 4^N)$ 키패드에 있는 알파벳 3개, 4개 공간 복잡도 : $O(3^N 4^N)$

42. Trapping Rain Water

layout: post title: 42. Trapping Rain Water category: leetcode date: 2025-01-27 12:27:00 +0900 description:

<https://leetcode.com/problems/trapping-rain-water/description/> img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

42. Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.



Example 1:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

Input: height = [4,2,0,3,2,5]

Output: 9

Solution (Two pointer)

```

class Solution {
public:
    int trap(vector<int>& height) {
        int n = height.size();
        int left = 0, right = n - 1;
        int leftMax = 0, rightMax = 0;
        int trappedWater = 0;

        while (left < right) {
            // 왼쪽 벽이 오른쪽 벽보다 낮은 경우
            if (height[left] < height[right]) {
                // 현재 왼쪽 벽이 왼쪽 최대 높이보다 높으면 업데이트
                if (height[left] >= leftMax) {
                    leftMax = height[left];
                } else {
                    // 아니면 물을 채움
                    trappedWater += (leftMax - height[left]);
                }
                left++;
            }
            // 오른쪽 벽이 왼쪽 벽보다 낮거나 같은 경우
            else {
                // 현재 오른쪽 벽이 오른쪽 최대 높이보다 높으면 업데이트
                if (height[right] >= rightMax) {
                    rightMax = height[right];
                } else {
                    // 아니면 물을 채움
                    trappedWater += (rightMax - height[right]);
                }
                right--;
            }
        }

        return trappedWater;
    }
};

```

Two pointer를 이용한 풀이, 시간 복잡도 : $O(N)$ 공간 복잡도 : $O(1)$

DP를 이용해서도 풀 수 있다.

Solution2 (DP)

```

class Solution {
public:
    int trap(vector<int>& height) {
        int n = height.size();
        if (n == 0) return 0; // 빈 배열이면 0 반환
    }
};

```



```
// 왼쪽 최대 높이를 저장하는 배열
vector<int> leftMax(n, 0);
// 오른쪽 최대 높이를 저장하는 배열
vector<int> rightMax(n, 0);

// 왼쪽 최대 높이 계산
leftMax[0] = height[0];
for (int i = 1; i < n; ++i) {
    leftMax[i] = max(leftMax[i - 1], height[i]);
}

// 오른쪽 최대 높이 계산
rightMax[n - 1] = height[n - 1];
for (int i = n - 2; i >= 0; --i) {
    rightMax[i] = max(rightMax[i + 1], height[i]);
}

// 각 인덱스에서 물의 양 계산
int totalWater = 0;
for (int i = 0; i < n; ++i) {
    int waterAtCurrent = min(leftMax[i], rightMax[i]) - height[i];
    if (waterAtCurrent > 0) {
        totalWater += waterAtCurrent;
    }
}

return totalWater;
};
```

시간 복잡도 : $O(N)$ 공간 복잡도 : $O(N)$

48. Rotate Image

layout: post title: 48. Rotate Image category: leetcode date: 2025-01-27 12:27:00 +0900 description: <https://leetcode.com/problems/rotate-image/description/> envType=company&envId=google&favoriteSlug=google-thirty-days img: leetcode.png # Add image post (optional) fig-caption: # Add figcaption (optional)

48. Rotate Image

You are given an $n \times n$ 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

Example 1:

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [[7,4,1],[8,5,2],[9,6,3]]

Example 2:

Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]

Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

Solution

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        //Tanspose
        int size = matrix[0].size();
        transpose(matrix, size);
        //reverse
        reverse(matrix, size);
    }
    void transpose(vector<vector<int>>& matrix, int size){
        for(int i=0; i<size; i++){
            for(int j=i; j<size; j++){
                swap(matrix[i][j],matrix[j][i]);
            }
        }
    }
    void reverse(vector<vector<int>>& matrix, int size){
        int middle = size/2;
        //middle을 기준으로 좌우대칭
        for(int i=0; i<size; i++){
            for(int j=0; j<middle; j++){
                swap(matrix[i][j],matrix[i][size-j-1]);
            }
        }
    }
};
```

시간 복잡도 $O(N^2)$ ($2 \cdot O(N^2)$) 공간 복잡도 $O(1)$

Hash table

Hash table

Hash table 구현

1. Linked list + hash code function
 - Worst case $O(n)$, normaly $O(1)$
 2. Balanced binary search tree
 - $O(\log N)$
 - Don't have to allocate array before use. (row space complexity)
-

Interview method

Interview method

1. 분석 능력
 - 문제 풀이에 도움을 받았는지
 - 최적화
 - 푸는데 걸린 시간
 - 디자인 및 설계
 2. 코딩 능력
 - 알고리즘
 - 가능한 예러에 대한 생각
 - 코드 스타일
 3. CS
 4. Experience
 5. Culture fit
-

Algorithm

Algorithm

박주홍 여기서 푸셈 이거 오랜만에 푸니까 재밌노 근데 머리 안 돌아감

Study process

Study process

1. Data structure
 - Array
 - static
 - dynamic
 - Linked list
 - single linked list
 - double linked list

- circle linked list
- insert, delete, search ...
- Stack
 - array based stack
 - linked list based stack
 - push, pop, peek ...
- Queue
 - array based queue
 - linked list based queue
 - circular queue
 - priority queue
 - enqueue, dequeue ...
- Hash table
 - hash function & collision
 - hash set, hash map

2. Basic algorithm

- Sort
 - bubble sort
 - select sort
 - insert sort
 - merge sort (Divide and conquer)
 - quick sort (Divide and conquer)
 - heap sort
- Search
 - linear search
 - binary search (Divide and conquer)

3. Intermediate algorithm

- Tree
 - binary tree
 - binary search tree (BST)
 - balanced binary search tree (AVL Tree, Red-Black Tree)
 - heap (max heap, min heap)
- Graph
 - adjacent matrix, adjacent list
 - DFS
 - BFS
 - dijkstra, Bellman-ford
 - minimum spanning tree(MST, Kruskal, Prim) (Greedy)
- Trie
 - insert, delete, search
- Union Find

4. Advanced algorithm

- DP
 - Memoization
 - Tabulation
 - Fibonacci, Back packing, Longest common subsequence(LCS)
 - Backtracking
 - N-Queen
 - Divide and conquer
 - Greedy
-

String

String

Ask about...

1. **ASCII** or **UNICODE**. (Normally use ASCII)
2. Upper/lower case handling.
3. Space or tab handling.

Technique

1. Solving the problem **backward**.

Matrix

Technique

1. Transpose / Reverse ...
2. If you have to approach many time for each element, consider using flag with other row or column. This technique reduce time complexity.

example below

```
// In MxN Matrix, if one element is 0, then change every value to 0 in same row
and column.
#include <iostream>
#include <vector>

void setZeroes(std::vector<std::vector<int>>& matrix) {
    int rows = matrix.size();
    int cols = matrix[0].size();

    bool firstRowZero = false;
    bool firstColZero = false;

    for (int j = 0; j < cols; ++j) {
        if (matrix[0][j] == 0) {
            firstRowZero = true;
```

```
        break;
    }
}

for (int i = 0; i < rows; ++i) {
    if (matrix[i][0] == 0) {
        firstColZero = true;
        break;
    }
}

for (int i = 1; i < rows; ++i) {
    for (int j = 1; j < cols; ++j) {
        if (matrix[i][j] == 0) {
            matrix[i][0] = 0;
            matrix[0][j] = 0;
        }
    }
}

for (int i = 1; i < rows; ++i) {
    if (matrix[i][0] == 0) {
        for (int j = 1; j < cols; ++j) {
            matrix[i][j] = 0;
        }
    }
}

for (int j = 1; j < cols; ++j) {
    if (matrix[0][j] == 0) {
        for (int i = 1; i < rows; ++i) {
            matrix[i][j] = 0;
        }
    }
}

if (firstRowZero) {
    for (int j = 0; j < cols; ++j) {
        matrix[0][j] = 0;
    }
}

if (firstColZero) {
    for (int i = 0; i < rows; ++i) {
        matrix[i][0] = 0;
    }
}
}
```

Bool

1. Use **bit vector** for reduce space complexity.
 - If you use **bool** instead of bit vector, the bool use 1byte(8bit), so it's space complexity is 8times more than bit vector.
- 2.

Permutation

If num1 and num2 have relationship with permutation....

1. sort() will help.
2. count element's frequency.

Palindrome

Every algorithm's minimum time complexity that solving palindrome is $O(N)$ because we should contact every single element.

Technique

1. Hash table
 - Count each element's frequency.
 - After loop, if one element's count is odd, then it's not palindrome.
 - Time complexity : $O(N)$
2. If the problem is determination problem, check the count number same time when approach element whether it's odd or not.
 - If you doing like this, you can get palindrome determination ASAP when for loop is done.
3. **Bit vector** (In case of string)

```
bool isPalindromePermutation(const std::string& s) {
    int bitVector = 0;
    for (char c : s) {
        if (std::isalpha(c)) { // Alphabet to bit vector
            int index = std::tolower(c) - 'a';
            bitVector ^= (1 << index); // Bit toggle with XOR
        }
    }
    //If
    return bitVector == 0 || (bitVector & (bitVector - 1)) == 0;
}
```

- bitVector is int because alphabet is only 26, so don't have to expand if string is ascii.
- First bitVector is 00000000 00000000 00000000 00000000 (in case of 32 bits env), then if for loop find 'b', then bitVector is now 00000000 00000000 00000000 00000010 because $\text{tolower}('b') - 'a' = 1$.
- After all, if bitVector have '1', then it's not palindrome.
- But, If input s is odd size, one '1' exist.

- So, this $(\text{bitVector} \& (\text{bitVector} - 1))$ will work in that case. (if this condition make you confuse, then just split case with $s.size()$ is odd or even. if $s.size()$ is odd, then check $\text{bitVector} == 1$.)
- Time complexity $O(N)$