Table of Contents

Binary Search

Binary Search

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

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

            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return -1;
    }
};
```

Sorting 되어있다는 가정하에 쓸모가있음.

중간지점 선택 후

1. target과 같은지 확인 (같다면 바로 return)

2. target보다 mid에서의 값이 작다면 left를 mid+1로 수정.
3. target보다 mid에서의 값이 크다면 right를 mid-1로 수정

삽입위치 찾기

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int left = 0, right = nums.size() - 1;
        int mid = 0;
        while (left <= right) {
            mid = left + (right - left) / 2;
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return left;
    }
};
```

```
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;
            }
        }
    }
```

Binary Search Tree

이진탐색트리인지 검증

```
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return validate(root, LONG_MIN, LONG_MAX);
    }

private:
    bool validate(TreeNode* node, long min, long max) {
        if (node == nullptr) {
            return true;
        }

        if (node->val <= min || node->val >= max) {
            return false;
        }

        return validate(node->left, min, node->val) && validate(node->right, node->val, max);
    }
};
```

전위순위 Pre order

이진탐색트리 회복

```
class Solution {
public:
    void recoverTree(TreeNode* root) {
        vector<TreeNode*> nodes;
        vector<int> values;
        inorderTraversal(root, nodes, values);
        sort(values.begin(), values.end());
        for (int i = 0; i < nodes.size(); i++) {
            nodes[i]->val = values[i];
        }
    }
```

```
private:
    void inorderTraversal(TreeNode* root, vector<TreeNode*>& nodes, vector<int>&
values) {
        if (root == nullptr) {
            return;
        }
        inorderTraversal(root->left, nodes, values);
        nodes.push_back(root);
        values.push_back(root->val);
        inorderTraversal(root->right, nodes, values);
    }
};
```

DP

`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.
3. 1 step + 1 step + 1 step
4. 1 step + 2 steps
5. 2 steps + 1 step

```
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];
    }
};
```

`Pascal's triangle'

```
class Solution {
public:
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> v;

        for (int i = 0; i < numRows; i++) {
            vector<int> temp(i + 1, 1);  // i + 1 크기로 벡터 초기화, 모든 값은 1로
설정

            // 중간 값을 계산
            for (int j = 1; j < i; j++) {
                temp[j] = v[i - 1][j - 1] + v[i - 1][j];
            }

            v.push_back(temp);  // 벡터에 temp 추가
        }

        return v;
    }
};
```

ETC

(32bits) INT_MAX = 2^31-1 INT_MIN = -2^31

함수 파라미터 적을 때, void inorder(TreeNode* root,vector& result) 여기서 &를 빼먹지말자. 참조연산자가 없으면 값만 들어간다.

기본 Struct

기본 Struct

```
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) {}
};
```

Reverse

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

Add Two Number

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* dummy = new ListNode();
        ListNode* current = dummy;
        int carry = 0;
        while (l1 || l2 || carry) {
            int sum = carry;
            if (l1) {
                sum += l1->val;
                l1 = l1->next;
            }
            if (l2) {
                sum += l2->val;
                l2 = l2->next;
            }
            carry = sum / 10;
            current->next = new ListNode(sum % 10);
            current = current->next;
        }
        return dummy->next;
    }
};
```

```
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) {
            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;
    }
};
```

중간을 기점으로 좌우 같으면 포인터를 확대

Time O(N^2) Space O(1)

---

permutation

permutation

```
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;
        }
    }
};
```

Priority Queue

Priority Queue

Input: ["KthLargest", "add", "add", "add", "add", "add"] [[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]] Output: [null, 4, 5, 5, 8, 8]

```
class KthLargest {
public:
    priority_queue<int> pq;  // 기본적으로 max-pq 사용
    int temp = 0;

    KthLargest(int k, vector<int>& nums) {
        temp = k;
        for (int num : nums) {
            add(num);
        }
    }

    int add(int val) {
        pq.push(-val); // 값을 음수로 변환하여 삽입 (최소 힙처럼 동작)
        // PQ에서는 sort할 필요없이 자동으로 정렬해준다. 기본적으로 작은값이 위로감

        // 힙의 크기가 k를 초과하면 가장 작은 값을 제거
        if (pq.size() > temp) {
            pq.pop();
        }

        // k번째로 큰 값 반환 (음수로 저장했으므로 다시 양수로 변환)
        return -pq.top();
    }
};
```

Problem)Combination_Sum

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: []

```
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]);
                int temp = target - candidates[i];
                calculation(candidates, temp, i); // `i`를 전달하여 중복 조합을 방지
                current.pop_back(); // 다음 조합을 위해 마지막 요소 제거
            }
        }
    }
};
```

Problem)Minimum Size Subarray Sum

209. Minimum Size Subarray Sum

Given an array of positive integers nums and a positive integer target, return the minimal length of a subarray whose sum is greater than or equal to target. If there is no such subarray, return 0 instead.

Example 1:

Input: target = 7, nums = [2,3,1,2,4,3] Output: 2 Explanation: The subarray [4,3] has the minimal length under the problem constraint. Example 2:

Input: target = 4, nums = [1,4,4] Output: 1 Example 3:

Input: target = 11, nums = [1,1,1,1,1,1,1,1] Output: 0

```
class Solution {
public:
    int minSubArrayLen(int target, vector<int>& nums) {
    int n = nums.size();
    int left = 0, sum = 0;
    int minLength = INT_MAX;

    for (int right = 0; right < n; ++right) {
        sum += nums[right];

        // Shrink the window as small as possible while the sum is >= target
        while (sum >= target) {
            minLength = min(minLength, right - left + 1);
            sum -= nums[left];
            ++left;
        }
    }

    return (minLength == INT_MAX) ? 0 : minLength;
}
};
```

sliding window

Problem)Number_of_islands

200. Number of Islands Solved Medium Topics Companies Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

Input: grid = [["1","1","1","1","0"], ["1","1","0","1","0"], ["1","1","0","0","0"], ["0","0","0","0","0"]] Output: 1
Example 2:

Input: grid = [["1","1","0","0","0"], ["1","1","0","0","0"], ["0","0","1","0","0"], ["0","0","0","1","1"]] Output: 3

```
class Solution {
public:
    int numIslands(vector<vector<char>>& grid) {
        if (grid.empty() || grid[0].empty()) return 0;

        int rows = grid.size();
        int cols = grid[0].size();
        int num_islands = 0;

        // 방향 벡터: 상, 하, 좌, 우
        vector<pair<int, int>> directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                if (grid[i][j] == '1') {
                    // 새로운 섬 발견
                    ++num_islands;
                    grid[i][j] = '0'; // 방문 표시 (땅을 물로 바꿈)

                    // BFS를 위한 큐 초기화
                    queue<pair<int, int>> q;
                    q.push({i, j});

                    while (!q.empty()) {
                        auto [cur_row, cur_col] = q.front();
                        q.pop();

                        // 현재 위치에서 상하좌우 탐색
                        for (auto [dr, dc] : directions) {
                            int new_row = cur_row + dr;
                            int new_col = cur_col + dc;

                            // 유효한 범위 내에 있고, 땅('1')인 경우
                            if (new_row >= 0 && new_row < rows &&
                                new_col >= 0 && new_col < cols &&
                                grid[new_row][new_col] == '1') {
                                grid[new_row][new_col] = '0'; // 방문 처리
                                q.push({new_row, new_col});
                            }
                        }
                    }
                }
            }
        }

        return num_islands;
    }
};
```

Problem)Rotting Oranges

994. Rotting Oranges

You are given an m x n grid where each cell can have one of three values:

0 representing an empty cell, 1 representing a fresh orange, or 2 representing a rotten orange. Every minute, any fresh orange that is 4-directionally adjacent to a rotten orange becomes rotten.

Return the minimum number of minutes that must elapse until no cell has a fresh orange. If this is impossible, return -1.

Example 1:

Input: grid = [[2,1,1],[1,1,0],[0,1,1]] Output: 4 Example 2:

Input: grid = [[2,1,1],[0,1,1],[1,0,1]] Output: -1 Explanation: The orange in the bottom left corner (row 2, column 0) is never rotten, because rotting only happens 4-directionally. Example 3:

Input: grid = [[0,2]] Output: 0 Explanation: Since there are already no fresh oranges at minute 0, the answer is just 0.

```
class Solution {
public:
    int orangesRotting(vector<vector<int>>& grid) {
        int rows = grid.size();
        int cols = grid[0].size();
        queue<pair<int, int>> q; // 좌표 저장용
        int freshOranges = 0; // 신선한 오렌지를 지표로 쓸거임
        for(int i=0; i<rows; i++){
            for(int j=0; j<cols; j++){
                if(grid[i][j]==2) q.push(make_pair(i,j));
                //시작부터 썩어있는 좌표를 queue안에 넣는다.
                //나중에 이 큐를 이용해서 초기 접근할거임.
            }
        }
        // 썩은 오렌지가 없고 신선한 오렌지가 없는 경우
        if (freshOranges == 0) return 0;
        vector<pair<int, int>> directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
        int minutes = 0;

        //여기까지가 초기 설정

        while (!q.empty()) {
            int size = q.size();
            bool rotted = false;

            for (int i = 0; i < size; ++i) {
                auto [x, y] = q.front();
                q.pop();

                // 4방향 탐색 // directions 이용해서 깔끔하게 접근하는 방법 외워두기
                for (auto [dx, dy] : directions) {
```

```
                    int nx = x + dx;
                    int ny = y + dy;

                    // 유효한 위치인지 확인하고, 신선한 오렌지를 썩게 만듦
                    if (nx >= 0 && ny >= 0 && nx < rows && ny < cols && grid[nx]
[ny] == 1) {
                        grid[nx][ny] = 2; // 썩음으로 변경
                        q.push({nx, ny});
                        freshOranges--;
                        rotted = true;
                    }
                }
            }

            // 만약 썩은 오렌지가 있었다면 시간 증가
            if (rotted) minutes++;
        }

        // 신선한 오렌지가 남아 있다면 -1 반환, 아니면 걸린 시간 반환
        return freshOranges == 0 ? minutes : -1;

    }
};
```

Problem)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

```
class Solution {
public:
    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];
}
};
```

Problem)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

```
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;
    }
};
```

기본 Struct 형태

기본 Struct 형태

```
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) {}
};
```

Invert Binary Tree

좌우 반전

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

```
    }
};
```

Inorder

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorder(root,result);
        return result;
    }
    void inorder(TreeNode* root,vector<int>& result){
        if(!root) return;
        inorder(root->left,result);
        result.push_back(root->val);
        inorder(root->right,result);
    }
};
```

Bianry tree

같은 트리인지 검증

```
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if (!p&& !q) return true;
        if (!p|| !q) return false;
        if (p->val != q->val) return false;

        return isSameTree(p->left,q->left) && isSameTree(p->right,q->right);
    }
};
```

같지 않을때 false 리턴해주는 부분 중요

대칭 검증

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if (!root) return true;
        return isMirror(root->left, root->right);
    }

    bool isMirror(TreeNode* left, TreeNode* right) {
```

```
        if (!left && !right) return true;
        if (!left || !right) return false;
        if (left->val != right->val) return false;
        return isMirror(left->left, right->right) && isMirror(left->right, right-
>left);
    }
};
```

마지막줄이 중요

최대 깊이 구하기

```
class Solution {
public:
int cnt =0;
    int maxDepth(TreeNode* root) {
        if (!root) return 0;
        return 1 + max(maxDepth(root->left), maxDepth(root->right));
    }
};
```