Algorithm and Problem Solving Quick Guide in C++

Data Structures, Algorithms and Coding Interview Problem Patterns in C++

rfdavid, 2025

rfdavid.com

INTRODUCTION

Motivation

The tech industry hiring standard is based on algorithm and data structure.

There are plenty of free resources available around algorithms and data structures. The purpose of this project is to be a quick guide where you can learn and review algorithms and data structures.

Some of the intended **key features:**

- Non-verbose, short-structured, and easy to follow descriptions
- Slide-based, practical for reviewing
- Free and open-source

⭐ **If you like, please add a star at *github.com/rfdavid/cpp-algo-cheatsheet***

Some Useful Links

Tech Interview Handbook
https://www.techinterviewhandbook.org
A very well-structured resource for interview preparation

Blind 75 Leetcode Questions
https://leetcode.com/discuss/general-discussion/460599/blind-75-leetcode-questions

Blind 75

- Blind 75 is a popular list of algorithm problems that intends to cover the main data structures and patterns.

- It is a curated list of 75 popular coding questions created by an ex-Meta Staff Engineer

Array

- ✓ Two Sum
- ✓ Contains Duplicate
- ✓ Product of Array Except Self
- ✓ Best Time to Buy and Sell Stock
- ✓ Maximum Subarray
- ✓ Maximum Product Subarray
- ✓ Find Minimum in Rotated Sorted Array
- ✓ Search in Rotated Sorted Array
- ✓ 3 Sum
- ✓ Container With Most Water

Binary

- ✓ Sum of Two Integers
- ✓ Number of 1 Bits
- ✓ Counting Bits
- ✓ Missing Number
- ✓ Reverse Bits

Dynamic Programming

- ✓ Climbing Stairs
- Coin Change
- Longest Increasing Subsequence
- ✓ Longest Common Subsequence
- Word Break
- Combination Sum
- House Robber
- House Robber II
- Decode Ways
- ✓ Unique Paths
- Jump Game

Matrix

- ✓ Set Matrix Zeroes
- ✓ Spiral Matrix
- ✓ Rotate Image
- Word Search

Blind 75

Tree

✓ Maximum Depth of Binary Tree

✓ Same Tree

✓ Invert/Flip Binary Tree

✓ Binary Tree Maximum Path Sum

✓ Binary Tree Level Order Traversal

✓ Serialize and Deserialize Binary Tree

✓ Subtree of Another Tree

Construct Binary Tree from Preorder and Inorder Traversal

Validate Binary Search Tree

✓ Kth Smallest Element in a Binary Search Tree

Lowest Common Ancestor of Binary Search Tree

Implement Trie (Prefix Tree)

Add and Search Word

Word Search II

Heap

✓ Top K Frequent Elements

Find Median from Data Stream

String

✓ Longest Substring Without Repeating Characters

✓ Longest Repeating Character Replacement

✓ Minimum Window Substring

✓ Valid Anagram

✓ Group Anagrams

✓ Valid Parentheses

✓ Valid Palindrome

Longest Palindromic Substring

Palindromic Substrings

Encode and Decode Strings ⭐

Linked List

✓ Reverse a Linked List

✓ Detect Cycle in a Linked List

✓ Merge Two Sorted Lists

✓ Merge K Sorted Lists

✓ Remove Nth Node From End Of List

✓ Reorder List

Graph

✓ Clone Graph

✓ Course Schedule

✓ Pacific Atlantic Water Flow

✓ Number of Islands

✓ Longest Consecutive Sequence

Alien Dictionary ⭐

✓ Graph Valid Tree ⭐

✓ Number of Connected Components In an Undirected Graph ⭐

Interval

✓ Insert Interval

✓ Merge Intervals

✓ Non-overlapping Intervals

✓ Meeting Rooms ⭐

Meeting Rooms II ⭐

⭐ = leetcode premium

Other problems

✓ Maximum Level Sum of a Binary Tree

✓ Minimum Number of Increments on Subarrays to Form a Target Array

✓ Leaf-Similar Trees

✓ Count Good Nodes in Binary Tree

Min Cost Climbing Stairs

Longest Palindromic Subsequence

✓ Minimum Cost for Tickets

✓ Webcrawler

✓ Network Delay Time

✓ Rotated Digits

⭐ = leetcode premium

Time Complexity

If the input gets bigger, how many steps does the algorithm take?

- Measure of how much the execution time of an algorithm grows relative to the size of its input (usually called n)
- Expressed in **Big-O notation** (e.g. O(1), O(n), O(n^2), etc) to describe the **upper bound** of how fast the algorithm's runtime grows
- Asymptotic notations

  Big-O (O) – Upper Bound (**Worst-case**): Describes the maximum amount of time/memory an algorithm could take.

  Theta Θ – Tight Bound (**Exact**): describes both the **upper** and **lower bound** (the **exact** growth rate)

  Omega (Ω) – Lower Bound (**Best-case**): describes the minimum time/space the algorithm needs.

Time Complexity

Examples

O(1)

Examples	Problems
Accessing an array element (`arr[i]`)	Hash table lookups
Swapping two variables	Checking if a number is even/odd
Stack/Queue `push` or `pop`	Returning first element of a list

O(log n)

Binary Search	Search in sorted array
Balanced BST insert/find (AVL, Red-Black Tree)	Find k-th smallest in BST
Finding floor/ceil in sorted array	Finding square root with binary search

O(n)

Linear Search	Maximum subarray sum (Kadane's algo)
Finding min/max in an array	Counting frequencies with hash map
Traversing linked list or array	One-pass string processing problems

Time Complexity

$O(n \log n)$

Merge Sort / Heap Sort	Sorting an array
Efficient algorithms for Closest Pair	Finding inversion count in array
Heapify operations	Kth largest element with heap

$O(n^2)$

Bubble Sort / Insertion Sort	Two Sum (brute force)
Checking all pairs in array	Longest Palindromic Substring (DP)
Floyd-Warshall algorithm	Edit Distance (DP)

$O(n^3)$

Matrix multiplication (naive)	Boolean matrix multiplication
DP on subsequences of length 3	Some DP path-finding problems
Floyd-Warshall for dense graphs	Counting triangles in graph

$O(2^n)$

Recursive Fibonacci (no memo)	Subset sum (brute force)
Backtracking for combinations/permutations	N-Queens
Traveling Salesman (brute force)	All subsets of array (power set)

Time Complexity

V = vertices (nodes)
E = edges

$O(n!)$

Generating all permutations	Traveling Salesman (brute force)
Brute-force anagram check	Word ladder with all transformations
Solving puzzles with all arrangements	Hamiltonian Path

$O(V + E)$

DFS / BFS (adjacency list)	DFS / BFS (adjacency list)

$O(E \log V)$

Dijkstra with priority queue	Dijkstra with priority queue

$O(VE)$

Bellman-Ford

$O(V^3)$

Floyd-Warshall

$O(E \log E)$

Kruskal's MST algorithm

Space Complexity

As the input size n grows, how much extra memory does the algorithm need to run?

- Measure of how much memory an algorithm uses relative to **input size**
- Expressed in **Big-O notation** (e.g. O(1), O(n), O(n^2), etc)
- It includes auxiliary space (extra memory used by the algorithm, not counting the input itself) and sometimes considers input space depending on the context
- Count only extra space needed **(exclude output)**
- The space complexity of a recursive tree traversal is **O(h),** where h is the height of the tree. This is because each recursive call adds a frame to the call stack, and in the worst case, the maximum stack depth is proportional to the tree's height

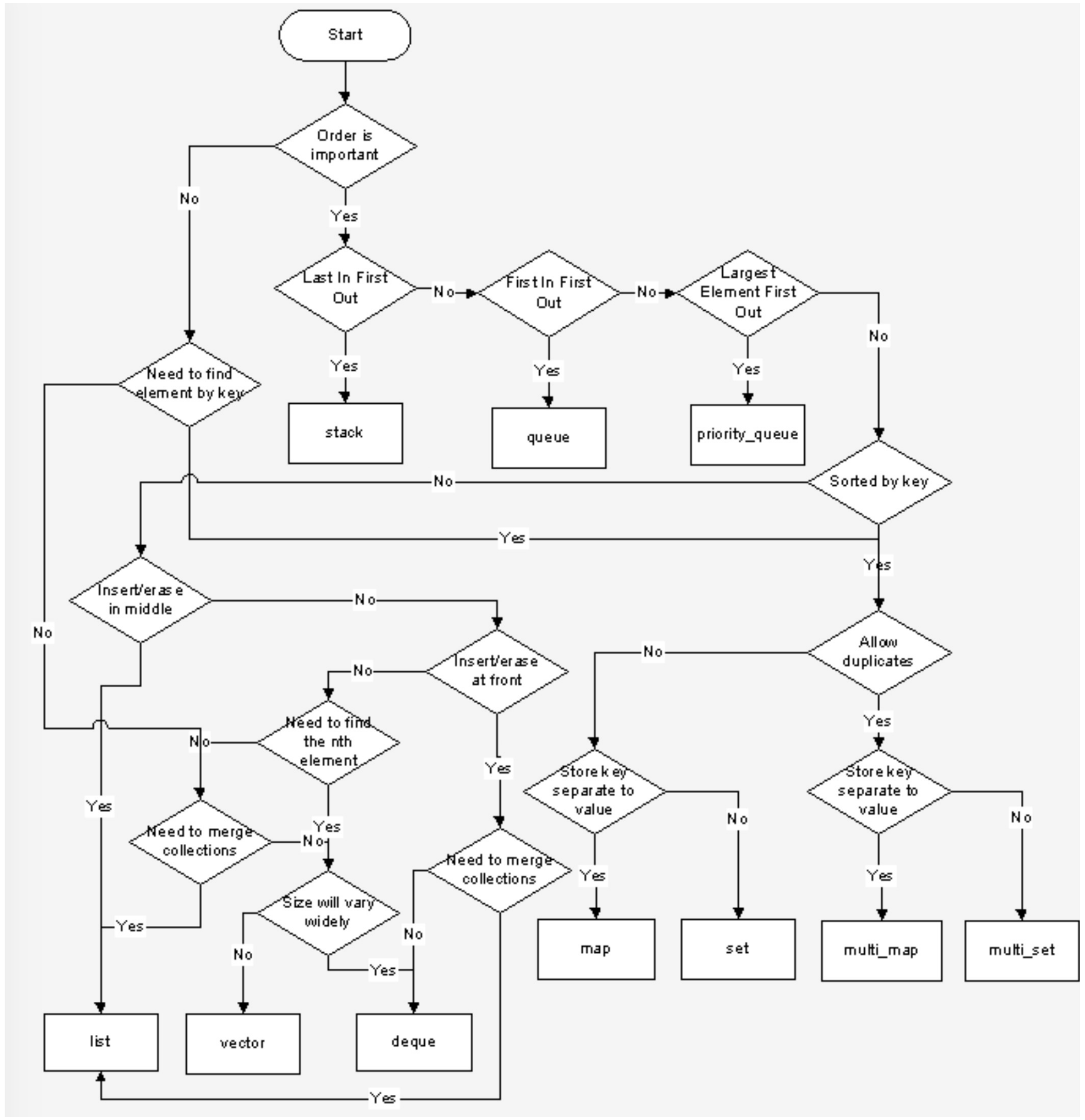
Space Complexity

Examples

Algorithm / Operation	Space Complexity	Explanation
Swap two integers	O(1)	Only uses constant space
Iterate through array and sum values	O(1)	No extra memory used besides accumulator
Store array copy	O(n)	Needs space to store the copied array
Recursive factorial (factorial(n))	O(n)	n stack frames in the call stack
Binary search (recursive)	O(log n)	Recursive depth is log(n) for sorted array
Binary search (iterative)	O(1)	No extra space beyond a few variables
Merge sort	O(n)	Needs temp arrays to merge subarrays
Quick sort (in-place)	O(log n)	Call stack for recursive calls
Depth-first search (recursive) in tree	O(h)	h = height of the tree (stack frames)
Breadth-first search (using queue)	O(n)	Stores all nodes at current level in queue
DP with full 2D table (e.g., LCS)	O(m*n)	Stores results of all subproblems
Optimized Fibonacci with two variables	O(1)	Only tracks last two results
Memoized Fibonacci (top-down DP)	O(n)	Memoization table + recursion stack
Using a hash map to count frequencies	O(n)	Stores one count per element
Storing all substrings of a string	O(n^2)	Total number of substrings is ~n^2
Adjacency list for graph with V nodes, E edges	O(V + E)	One list per node, total edges stored

DATA STRUCTURES IN C++

Data Structure Decision Diagram



Note: I don't have the source of this diagram. If you know it, please drop me a msg so I can add it here.

- The following diagram gives you the direction to which data structure to use in C++ according to the problem you are trying to solve

Arrays

- std::vector is a sequence container that encapsulates dynamic sized arrays*

*https://en.cppreference.com/w/cpp/container/vector

Linked List

- desc

Stack

- desc

*https://en.cppreference.com/w/cpp/container/vector

Queue

- desc

*https://en.cppreference.com/w/cpp/container/vector

Heap

- desc

Hash Table

- desc

Tree

- desc

ARRAY

Arrays

- **Memory layout**: hold values in a **contiguous** block of memory.
- **Fixed Size**: the size of an array is defined when it is created and cannot be changed. However, high-level languages have different implementations, making it dynamic.
- **Homogeneous elements:** all elements are of the same data type (int, float, char…)
- **Efficiency**: accessing elements by index is very efficient *O(1)*, since each index maps directly to a memory location. Also, range scans benefit from CPU cache lines since arrays are stored in contiguous blocks of memory.

Problem – 1. Two Sum

Easy

LeetCode leetcode.com/problems/two-sum

Problem

- Given an **`array`** of numbers and a **`target`**, example: **`array [2,7,11,15]`** and **`target 9`**
- Return indices of two numbers where they add up to **`target`**
- **Output**: [0,1]

```
array[0] + array[1] = 2 + 7 = 9
```

Solution – 1. Two Sum

Easy

LeetCode leetcode.com/problems/two-sum

Solution

- Iterative over each number in the array
- Calculate the difference between target and each number, example:

  `array[0] = 2, target 9, then 9 - 2 = 7`
- Now we know we need the number **7** to sum up to **9**
- Check in a *hashmap* if we have 7 in some part of the array

  `hash[7] exists?`
- If yes, return the current index and the index of 7
- If not, store the index of the current number in the hashmap for future evaluation

  `hash[2] = 0`

Code – 1. Two Sum

Easy

LeetCode leetcode.com/problems/two-sum

Code

Time: **O(n)** Space: **O(n)**

```
vector<int> twoSum(vector<int>& nums, int target) {
    std::unordered_map<int, int> numMap;
    // n being the size of nums
    for (int i = 0; i < nums.size(); i++) {
        // current number of the array
        int number = nums[i];
        int diff = target - number;

        // check if the difference is in some part of the array
        // by using a hashmap
        if (numMap.find(diff) != numMap.end()) {
            return { numMap[diff], i};
        }

        // register the current number index
        numMap[number] = i;
    }
    // no matches
    return {};
}
```

Problem – 217. Contains Duplicate

Easy

LeetCode leetcode.com/problems/contains-duplicate

Problem

- You are given an array of numbers
- Return any value that appears at least twice

Solution

- Loop through the array
- Check if the value is in a hash table
- Return **true** if the value exist
- The problem requires at least twice, but one modification may be having a specific count

Code – 217. Contains Duplicate

Easy

LeetCode leetcode.com/problems/contains-duplicate

Code

Time: O(n) Space: O(n)

```
bool containsDuplicate(vector<int>& nums) {
    unordered_map<int, int> seen;
    for (int i = 0; i < nums.size(); ++i) {
        if (seen[nums[i]] == 1) {
            return true;
        }
        seen[nums[i]]++;
    }
    return false;
}
```

Another solution (less flexible)

```
bool containsDuplicate(vector<int>& nums) {
    unordered_map<int, bool> seen;
    for (const auto& num : nums) {
        if (seen[num]) {
            return true;
        }
        seen[num] = true;
    }
    return false;
}
```

Problem – 238. Product of Array Except Self

Medium

LeetCode leetcode.com/problems/product-of-array-except-self

Problem Statement

- You are given an integer array **nums**
- Return another array where each element is multiplied by all the elements except itself
- Example:

```
nums = [14,2,5,99]

nums[0] = 2 * 5 * 99 (all except 14)

nums[1] = 14 * 5 * 99 (all except 2)

nums[2] = 14 * 2 * 99

nums[3] = 14 * 2 * 5
```

Solution – 238. Product of Array Except Self

Medium

LeetCode leetcode.com/problems/product-of-array-except-self

Solution

- Go over the array once and calculate the product of the left side. Example:

```
nums = [14,2,5]
left[0] = 1  (think of multiplying all elements before 14, so 1 because there is none)
left[1] = 14 (all elements from the left multiplied, except 2)
left[2] = 14 * 2 = 28 (all elements from the left multiplied, except 5)
left = [1, 14, 28]
```

- Using the same logic, do the same calculation but starting from the right

```
right[2] = 1 (no elements after 5)
right[1] = 5 (only 5 after 2)
right[0] = 2 * 5 = 10
right = [10, 5, 1]
```

- Multiply each element from left and right:

```
left ⊙ right = [1, 14, 28] ⊙ [10,5,1] = [10, 70, 28]
```

Code – 238. Product of Array Except Self

Medium

LeetCode leetcode.com/problems/product-of-array-except-self

Code Time: **O(n)** Space: **O(n)**

```cpp
vector<int> productExceptSelf(vector<int>& nums) {
    int n = nums.size();
    vector<int> output(n, 1);
    vector<int> right(n, 1);

    // calculate left first
    for (int i = 1; i < n; ++i) {
        output[i] = nums[i - 1] * output[i - 1];
    }
    // calculate right
    for (int i = n - 1; i >= 0; --i) {
        right[i] = nums[i + 1] * right[i + 1];
        output[i] = output[i] * right[i];
    }

    /* or you can save some space using this logic,
      although I don't find it as intuitive as the previous one
    int right = 1;
    for (int i = n - 1; i >= 0; --i) {
        output[i] *= right;
        right *= nums[i];
    */

    return output;
}
```

Problem – 53. Maximum Subarray

Medium

LeetCode leetcode.com/problems/maximum-subarray

Problem

- You are given an array nums
- Find the subarray with the largest sum
- **Example:**

```
nums = [-2,1,-3,4,-1,2,1,-5,4]

output = 6
```

The subarray `[4,-1,2,1]` has the largest sum 6.

Problem – 53. Maximum Subarray

Medium

LeetCode leetcode.com/problems/maximum-subarray

Solution

- Use Kadane's algorithm to find the maximum sum of a contiguous subarray in linear time
- Core idea:

  at each index, either:

  1. start a new subarray at **nums[i]** or

  2. extend the current one by adding **nums[i]**

Arrays – Kadane's algorithm

- Kadane's algorithm is a dynamic programming algorithm to solve **maximum subarray sum**
- At every **index i**:

  start a new subarray at **i**

  extend the previous subarray to include **array[i]**
- **Algorithm**

  1. Initialize:

```
int maxSoFar = array[0];
int maxEndingHere = array[0];
```

  2. Loop through the array

```
for (int i = 1; i < array.size(); ++i) {
    maxEndingHere = max(array[i], maxEndingHere + array[i]);
    maxSoFar = max(maxSoFar, maxEndingHere);
}
```

  3. Return maxSoFar;

Problem – 53. Maximum Subarray

Medium

LeetCode leetcode.com/problems/maximum-subarray

Code

Time: **O(n)** Space: **O(1)**

```
int maxSubArray(vector<int>& nums) {
    int maxSum = nums[0];
    int currentSum = nums[0];
    for (int i = 1; i < nums.size(); ++i) {
        currentSum = max(nums[i], currentSum + nums[i]);
        maxSum = max(maxSum, currentSum);
    }
    return maxSum;
}
```

Problem – 152. Maximum Product Subarray

Medium

LeetCode leetcode.com/problems/maximum-product-subarray

Problem

- You are given an array **nums**
- Find a subarray that has the largest product and return the product
- The array may contain negative numbers
- **Example:**

```
nums = [2, 3, -2, 4]
output = 6
```

`[2,3]` has the largest `6 (2 * 3)`

Problem – 152. Maximum Product Subarray

Medium

LeetCode leetcode.com/problems/maximum-product-subarray

Solution

- Use a modified version of Kadane's algorithm
- Keep track of the minimum and maximum product
- Once the current number is negative, swap minimum product with maximum product
- Check the largest product between maximum product and the final result

Problem – 152. Maximum Product Subarray

Medium

LeetCode leetcode.com/problems/maximum-product-subarray

Code

Time: **O(n)** Space: **O(1)**

```
 int maxProduct(vector<int>& nums) {
    int result = nums[0];
    int maxProd = nums[0];
    int minProd = nums[0];
    for (int i = 1; i < nums.size(); ++i) {
        if (nums[i] < 0) {
            swap(minProd, maxProd);
        }

        minProd = min(nums[i], nums[i] * minProd); // -2
        maxProd = max(nums[i], nums[i] * maxProd); // -30

        result = max(result, maxProd);
    }
    return result;
}
```

Problem – 153. Find Minimum in Rotated Sorted Array

Medium

LeetCode leetcode.com/problems/find-minimum-in-rotated-sorted-array

Problem

- You are given a sorted array but “rotated”
- Rotated means the elements are displaced in order
- Return the **minimum element**
- **Example:**

  `nums = [3,4,5,1,2]`

  `output = 1` (minimum element)

Problem – 153. Find Minimum in Rotated Sorted Array

Medium

LeetCode leetcode.com/problems/find-minimum-in-rotated-sorted-array

Solution

- Perform an adapted binary search
- Example:

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

  left = 3, **mid = 5, right =2**

  You find mid (5), but have to go right, so adjust left:

```
if (mid > right)
   left = mid + 1
else
   right = mid
```

Problem – 153. Find Minimum in Rotated Sorted Array

Medium

LeetCode leetcode.com/problems/find-minimum-in-rotated-sorted-array

Code

Time: **O(log n)** Space: **O(1)**

```
int findMin(vector<int>& nums) {
    int left = 0;
    int right = nums.size() - 1;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] > nums[right]) {
            left = mid + 1;
        } else {
            right = mid;
        }
    }
    return nums[left];
}
```

Problem - Best Time to Buy and Sell Stock

Easy

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock

Problem Statement

- You are given an integer **array** of stock prices
- Choose a **price[i]** to buy and **price[i]** to sell where you achieve maximum profits
- **Example:**

```
prices = [9, 1, 3, 4]
```

- **Output**: [1,3]

```
array[3] - array[1] = 4 + 1 = 3
```

Solution - Best Time to Buy and Sell Stock

Easy

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock

Solution

- Initialize **profit = 0**
- Initialize **lowestBuyPrice = prices[0]**
- Loop through the prices
- Track the lowest buy price → **min(lowestBuyPrice, prices[i])**
- Check if selling "today" will make the maximum profit and update profit:

  max(prices[i] – buy > profit, profit)
- Update profit

  max(prices[i] - buy

Code - Best Time to Buy and Sell Stock

Easy

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock

Code (simplified)

Time: **O(n)** Space: **O(1)**

```
int maxProfit(vector<int>& prices) {
    int profit = 0;
    int buy = prices[0];
    for (auto i = 1; i < prices.size(); i++) {
        buy = min(buy, prices[i]);
        profit = max(profit, prices[i] - buy)
    }
    return profit;
}
```

Code - Best Time to Buy and Sell Stock

Easy

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock

Code (optimized)

Time: **O(n)** Space: **O(1)**

- Same logic, but with better branch prediction and less computation

```
int maxProfit(vector<int>& prices) {
    int profit = 0;
    int buy = prices[0];
    for (auto 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;
}
```

Problem - Best Time to Buy and Sell Stock II

Medium

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-ii

Problem

- You are given an integer **array** of stock prices
- Choose a **price[i]** to buy and **price[i]** to sell where you achieve maximum profits
- You can buy/sell **multiple times**, but only hold **at most one** transaction at a time
- Output is the **maximum profits**
- **Example:**

  `prices = [9, 1, 3, 4]`

  Output: 2 + 1 = 3

  ```
  buy (price = 1), sell (price = 3), profit = 2
  buy (price = 3), sell (price = 4), profit = 1
  ```

Solution - Best Time to Buy and Sell Stock II

Medium

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-ii

Solution

- Loop through the array starting from index 1
- If current **price[i]** is lower than previous **price[i - 1]**, buy and sell
- **Example:**

```
prices = [1, 8, 4]      prices[0] = 1, prices[1] = 8, prices[2] = 4
prices[0] < prices[1] → true, profit = 8 - 1 = 7
prices[2] < prices[1] → false, do nothing
```

Code - Best Time to Buy and Sell Stock II

Medium

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-ii

Code

Time: **O(n)** Space: **O(1)**

```
int maxProfit(vector<int>& prices) {
    int profit = 0;
    for (int i = 1; i < prices.size(); ++i) {
        if (prices[i] > prices[i-1]) {
            profit += prices[i] - prices[i - 1];
        }
    }
    return profit;
}
```

Problem - Best Time to Buy and Sell Stock IV

Hard

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-iv

Problem Statement

- ...

Solution - Best Time to Buy and Sell Stock IV

Hard

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-iv

Solution

- ...

Code - Best Time to Buy and Sell Stock IV

Hard

LeetCode leetcode.com/problems/best-time-to-buy-and-sell-stock-iv

Code (simplified)

Time: **O(n)** Space: **O(n)**

```
int maxProfit(vector<int>& prices) {
    int profit = 0;
    int buy = prices[0];
    for (auto i = 1; i < prices.size(); i++) {
        buy = min(buy, prices[i]);
        profit = max(profit, prices[i] - buy)
    }
    return profit;
}
```

Problem – 33. Search in Rotated Sorted Array

Medium

LeetCode leetcode.com/problems/search-in-rotated-sorted-array

Problem

- Variation of *Find Minimum in Sorted Rotated Array problem*
- You are given a sorted **array** but “rotated” and a target number **n**
- Rotated means the elements are displaced in order
- Search the number **n** and return its index
- **Example**

  `nums = [4,5,6,7,0,1,2], target = 0`

  Output: 4

  4 is the index where the target number **0** is located

Problem – 33. Search in Rotated Sorted Array

Medium

LeetCode leetcode.com/problems/search-in-rotated-sorted-array

Solution

- Perform a binary search with some modification
- One side is always sorted, so find which side (left or right)
- Check if the target is in the range of the sorted side and adjust mid

  Example:

  [2,4,5,6,7,0,1] target = 0

  1. Find mid (6)
  2. Find the sorted side (left) = [2,4,5]
  3. Check if your target is in this side. Is **target** between 2 and 5?
  4. Adjust mid to search at the other side if not, otherwise continue searching at the same side

Problem – 33. Search in Rotated Sorted Array

Medium

LeetCode leetcode.com/problems/search-in-rotated-sorted-array

Code

Time: **O(log n)** Space: **O(1)**

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

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) return mid;

        // figure it out the sorted side
        if (nums[mid] >= nums[0]) {
            // left side is sorted
            // is target within this range?
            if (target >= nums[0] && target < nums[mid]) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        } else {
            // right side is sorted
            // is target within this range?
            if (target <= nums[right] && target > nums[mid]) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
    }
    return -1;
}
```

Problem – 15. 3Sum

Medium

LeetCode leetcode.com/problems/3sum

Problem

- You are given an array of integer **nums**
- Find distinct triples that the final sum is equal to zero
- **Example**:

  `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]`.

Problem – 15. 3Sum

Medium

LeetCode leetcode.com/problems/3sum

Solution

- Use three pointers: **i, j** and **k**
- Sort the array. This is necessary to move the pointers **j** and **k**
- Pointer **i** starts at the beginning the array
- Pointer **j** starts at i + 1 (second position)
- Pointer **k** starts at the end of the array
- Pointer **i** always move forward until the end of the array
- For each value of **i,** j and k will move either forward or backward, depending on the results of the sum
- Once find a sum == 0, add to a set to guarantee no duplicates

Problem – 15. 3Sum

Medium

LeetCode leetcode.com/problems/3sum

Code

Time: **O(n^2 log n)** Space: **O(n^2)**

```
vector<vector<int>> threeSum(vector<int>& nums) {
    set<vector<int>> triplets;
    sort(nums.begin(), nums.end());
    for (int i = 0; i < nums.size() - 2; ++i) {
        int j = i + 1;
        int k = nums.size() - 1;
        // it is a solution
        while (j < k) {
            int sum = nums[i] + nums[j] + nums[k];
            if (sum == 0) {
                triplets.insert({nums[i], nums[j], nums[k]});
                j++;
                k--;
            }
            if (sum < 0) {
                j++;
            } else {
                k--;
            }
        }
    }
    vector<vector<int>> result;
    // convert the solutions to the expected return
    for (const auto& t : triplets) {
        result.push_back(t);
    }

    return result;
}
```

Problem – 11. Container With Most Water

Medium

LeetCode leetcode.com/problems/container-with-most-water

Problem Statement

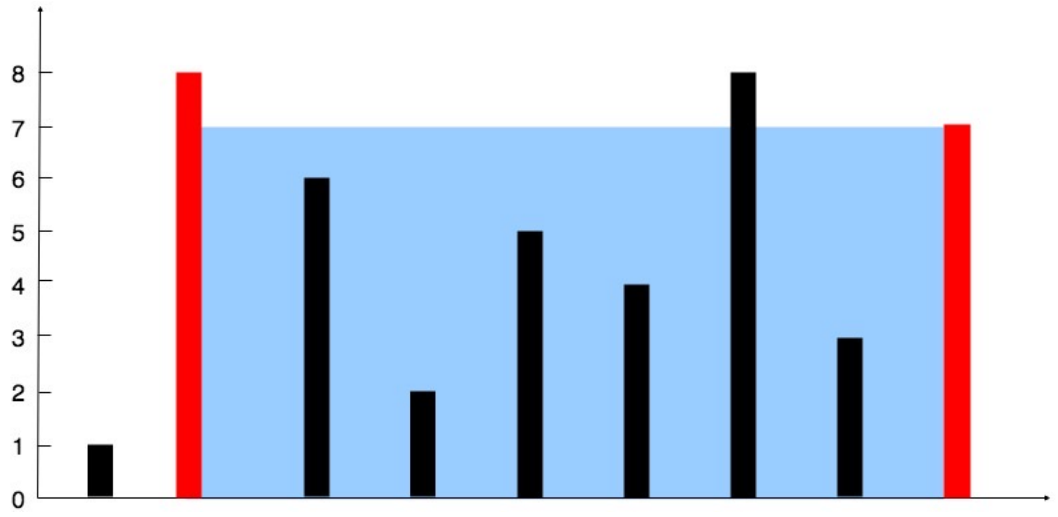
- You are given an integer array **height**
- Find two lines that together with x-axis form a container with most water
- Example:

  Input:

  height = [1,8,6,2,5,4,8,3,7]

  Output:

  49



Solution – Container With Most Water

Medium

LeetCode leetcode.com/problems/container-with-most-water

Solution

- Initialize the maximum area **maxArea = 0**
- Initialize two pointers, **left = 0** and **right = height.size – 1**
- Loop while pointer **left < right**
- Calculate the area:

  area = min(height[left], height[right]) * (right – left)
- Update the global maximum area:

  maxArea = max(maxArea, area)
- Move the smallest pointer (increment **left** or decrement **right**)
- Return **maxArea**

Problem – 11. Container With Most Water

Medium

LeetCode leetcode.com/problems/container-with-most-water

Code

Time: **O(n)** Space: **O(1)**

```cpp
int maxArea(vector<int>& height) {
    // left and right = positions
    int left = 0;
    int right = height.size() - 1;
    int maxArea = 0;
    while (left < right) {
        int area = min(height[left], height[right]) * (right - left);
        maxArea = max(maxArea, area);
        // adjust left and right based on 'greedy' algorithm
        // move from the lowest height
        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }
    return maxArea;
}
```

STRING

Problem – 3. Longest Substring Without Repeating Characters

Medium

LeetCode leetcode.com/problems/longest-substring-without-repeating-characters

Problem Statement

- You are given a string and the goal is to find the longest substring without repeating characters
- **Example**

  Input: "abcdb"

  Output: 4 (abcd since "b" is repeated)

Solution – 3. Longest Substring Without Repeating Characters

Medium

LeetCode leetcode.com/problems/longest-substring-without-repeating-characters

Solution

- Use sliding window algorithm (left and right)
- Loop through the string
- Try to find if the current character is already added by using unordered set or bitmap
- If added, remove from the set alongside with others using left pointer
- If not, add to the unordered set or bitmap
- Maximum length will be right – left + 1

Example – 3. Longest Substring Without Repeating Characters

Medium

LeetCode leetcode.com/problems/longest-substring-without-repeating-characters

Example

- String: abcbd. Our goal is to return 3 (**abc**bd)
- Initialize `maxLength = 0`
- Loop through the string

Iteration 1: `left = 0, right = 0, string[left] = 'a',`

`bitmap = ['a']` ('a' is not in bitmap, add), `maxLength = max(maxLength, right – left + 1) = 1`

Iteration 2: `left = 0, right = 1, string[right] = 'b'`

`bitmap = ['a','b'], maxLength = 2`

Iteration 3: `left = 0, right = 2, string[right] = 'c'`

`bitmap = ['a','b', 'c'], maxLength = 3`

Iteration 4: `left = 0, right = 3, string[right] = 'b'`

`bitmap = ['a','b','c','b']`

'b' is already in the bitmap. start "clearing" the character using left:

Iteration 4a: `left = 0, string[left] = 'a'` is different from 'b', so remove 'a'

`bitmap = ['b', 'c','b']`

Iteration 4b: `left = 1, string[left] = 'b'` is the same as the repeated one, remove

`bitmap = ['c','b']`

Iteration 5: `left = 1, right = 4, string[right] = 'd'`

`bitmap = ['c','b','d']`

Code – 3. Longest Substring Without Repeating Characters

Medium

Code (unordered_set)

- Use unordered_set when question requires unicode chars

```
int lengthOfLongestSubstring(string s) {
    int maxLength = 0;
    int left = 0, right = 0;
    // track the seen characters
    unordered_set<char> seen;
    for (right = 0; right < s.size(); ++right) {
        char currentChar = s[right];
        // if currentChar is in the set, clean
        // the character and everything from left of it
        // basically, reset the longest substring
        while (seen.count(currentChar)) {
            char c = s[left];
            seen.erase(c);
            left++;
        }
        // insert the current read character
        seen.insert(currentChar);
        // set max length
        maxLength = max(maxLength, right - left + 1);
    }
    return maxLength;
}
```

Code – 3. Longest Substring Without Repeating Characters

Medium

Code (bitmap)

- Using bitset: create a bitmask with 128 bits where each bit represent a character
- Optimal solution for ASCII since ASCII size is 127 characters
- Unicode / UTF-8 can represent over 1.1 million characters, so use **unordered_set** approach instead

```
int lengthOfLongestSubstring(string s) {
    std::bitset<128> bitmask;
    uint32_t left = 0;
    uint32_t maxLength = 0;

    for (uint32_t right = 0; right < s.length(); ++right) {
        uint32_t bitIndex = s[right];
        // if char is already in the bitmask, move left until we reset the bits
        while (bitmask.test(bitIndex)) {
            bitmask.reset(s[left]);
            ++left;
        }

        bitmask.set(bitIndex);
        maxLength = std::max(maxLength, right - left + 1);
    }
    return maxLength;
}
```

Problem – 424. Longest Repeating Character Replacement

Medium

LeetCode leetcode.com/problems/longest-repeating-character-replacement

Problem

- You are given a **string s** and an **integer k**
- You can replace one character by any other uppercase English character **k** times
- Return the longest substring with the same character
- **Example**:

  Input:

  ```
  s = "ABAB", k = 2
  ```

  Output: 4

  Replace the two 'A's with two 'B's or vice versa.

Problem – 424. Longest Repeating Character Replacement

Medium

LeetCode leetcode.com/problems/longest-repeating-character-replacement

Solution

- Start with two pointers: left and right
- Keep track of the frequencies of each letter in a **vector<int>** since we know there are 26 characters
- Initialize **maxFreq** to keep track of the letter with maximum frequency
- Initialize **maxLength** to keep track of the maximum substring
- Go over the string, and for each iteration:

  - calculate the windowSize

  - calculate the maximum frequency

  - check how many replacements is needed. That is, windowSize – maxFreq

  - if no replace can be done (k < replaces) then move left pointer to the right

Problem – 424. Longest Repeating Character Replacement

Medium

LeetCode leetcode.com/problems/longest-repeating-character-replacement

Code

Time: **O(n)** Space: **O(1)**

```cpp
int characterReplacement(string s, int k) {
    int left = 0;
    int maxLength = 0;
    int maxFreq = 0;
    vector<int> freq(26, 0);
    for (int right = 0; right < s.size(); ++right) {
        int index = s[right] - 'A';
        int windowSize = right - left + 1;
        // keep track of the frequencies
        freq[index]++;
        maxFreq = max(maxFreq, freq[index]);

        // check if the subwindow need to change
        int needReplace = windowSize - maxFreq;
        if (k < needReplace) {
            // need to move sub window
            int leftIndex = s[left] - 'A';
            freq[leftIndex]--;
            left++;
            windowSize = right - left + 1;
        }
        maxLength = max(maxLength, windowSize);
    }
    return maxLength;
}
```

Problem – 76. Maximum Window Substring

Hard

LeetCode leetcode.com/problems/minimum-window-substring

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 242. Valid Anagram

Easy

LeetCode leetcode.com/problems/valid-anagram

Problem

- You are given two strings **s** and **t**
- Return true if **t** is an anagram of **s**
- **Example:**

  ```
  t = word
  ```

  ```
  s = dwor
  ```

  Output: true

  both have the same number of same characters

Problem – 242. Valid Anagram

Easy

LeetCode leetcode.com/problems/valid-anagram

Solution

- Initialize a vector of integers to keep track of the count of each letter
- Loop over **s** and increase the count of each character found
- Then, loop over **t** and decrease the count of each character found
- Finally, loop over the vector and if there is one count greater than 0, return false

Problem – 242. Valid Anagram

Easy

LeetCode leetcode.com/problems/valid-anagram

Code

Time: **O(n)** Space: **O(1)**

```cpp
bool isAnagram(string s, string t) {
    // count the number of characters in 's', store in a vector
    // go over the vector and check if it's empty
    vector<int> letters(26);
    for (const auto& c : s) {
        letters[c - 'a']++;
    }
    for (const auto& c : t) {
        letters[c - 'a']--;
    }
    for (const auto& c : letters) {
        if (c != 0) return false;
    }
    return true;
}
```

Problem – 49. Group Anagrams

Medium

LeetCode leetcode.com/problems/group-anagrams

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- …

Problem – Valid Parentheses

Easy

LeetCode leetcode.com/problems/valid-parentheses

Problem Statement

- You are given a string containing only the characters **'(', ')', '{', '}', '[' and ‘]’**
- A valid input have closed brackets by its own type
- **Example**

  ()[]{} → valid

  []{}(→ invalid

  {()} → valid

Solution – Valid Parentheses

Easy

LeetCode leetcode.com/problems/valid-parentheses

Solution

- Loop through the string
- If **open** brackets **([{** push to a stack
- If **closed** brackets:

  pop the last added bracket

  check if the **closed** bracket corresponds to the **popped** bracket

  if not, return false
- after the loop, **return true** if the **size** of the stack is empty (all brackets closed)

Code – Valid Parentheses

Easy

LeetCode leetcode.com/problems/valid-parentheses

Code

Time: **O(n)** Space: **O(n)**

```
bool isValid(string s) {
    // stack (LIFO)
    std::stack<char> brackets;
    // O(n)
    for (int i = 0; i < s.size(); ++i) {
        char bracket = s[i];
        if (bracket == '(' || bracket == '[' || bracket == '{') {
            brackets.push(bracket);
        } else {
            if (brackets.size() == 0) return false;
            char lastBracket = brackets.top();
            if (bracket == ')' && lastBracket != '(') return false;
            if (bracket == '}' && lastBracket != '{') return false;
            if (bracket == ']' && lastBracket != '[') return false;
            brackets.pop();
        }
    }
    // all brackets must be closed
    return brackets.size() == 0;
}
```

Problem – 125. Valid Palindrome

Easy

LeetCode leetcode.com/problems/valid-palindrome

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- …

Problem – Minimum Number of Increments on Subarrays

Hard

LeetCode leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array

Problem Statement

- You are given an array of integers initialized with zeros (e.g. **[0,0,0,0]**)
- The goal is to reach some target (e.g. **[1, 2, 2, 3]**)
- The valid operations is to increment a subarray by one
- The output is the total number of operations

  In this case:

  [1,1,1,1] → increment the subarray starting from 0 to total size

  [1,2,2,2] → increment the subarray starting from 1 to total size

  [1,2,2,3] → increment the subarray starting and ending from the last element

  Output: 3 (total number of operations)

Solution – Minimum Number of Increments on Subarrays

Hard

LeetCode leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array

Solution

- Take this example:

  `target = [1000, 1, 1000]`

- Initialize total number of operations `totalOp = target[0] = 1000`
- Loop through the array, compare the first element with the previous:

  `target[1] > target[0] → 1 > 1000` → do nothing, totalOp is still 1000

- `target[2] > target[1] → 1000 > 1` → add the difference to totalOp:

  difference = 1000 - 1 = 999

  totalOp = 1000 + 999 = 1999

- This is the number of operations needed, equivalent to:

  - add 1 to each element: [1,1,1]

  - add 999 to the subarray [0,0]

  - add 999 to the subarray [2,2]

Code – Minimum Number of Increments on Subarrays

Hard

LeetCode leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array

Code

```
int minNumberOperations(vector<int>& target) {
    int totalOp = target[0];
    for (int i = 1; i < target.size(); ++i) {
        // can't reuse
        if (target[i - 1] < target[i]) {
            totalOp += target[i] - target[i - 1];
        }
    }
    return totalOp;
}
```

Code (2) – Minimum Number of Increments on Subarrays

Hard

LeetCode leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array

Code (optimized)

```
int minNumberOperations(vector<int>& target) {
    return target[0] +
        inner_product(target.begin() + 1, target.end(),
                target.begin(), 0,
                plus<int>(),
                [](int curr, int prev) { return max(curr - prev, 0); });
}
```

Problem – 788. Rotated Digits

Medium

LeetCode leetcode.com/problems/rotated-digits

Problem

- You are a given a number **n**
- From the range between 1 to n, find “good” numbers
- A good number must meet **2 requirements:**

  1. Be still valid after flipping: You physically “rotate” this number by 180 degrees, flip the number upside-down 2. The number can be either valid or invalid. For example, flipping **8** is still **8**, flipping **6** becomes **9**, but flipping **3**, becomes **ɛ** which is invalid.

  2. Be a different digit after flipping. If you flip **1,** it is still a valid number but it is the same number (1), so it is not good. However, **16** is valid because it becomes a different number: **19**
- Return the the number of good numbers between **1** and **n**

Problem – 788. Rotated Digits

Medium

LeetCode leetcode.com/problems/rotated-digits

Solution

- The simplest and readable approach:
- Create a function to check if a number is good or not
- Go over the range (1,n) and check every number. If it is good, count as a valid
- Inside the function to check:
- Extract digit by digit from the number (digit = num % 10)
- Check if the digit is valid (a.k.a "flippable"). In other words, return false if it is 3, 4 or 7.
- Now check the second condition (same number). So keep a bool "changed", if you find a number that "changes", mark changed as true. The numbers are 2, 5, 6 and 9, since when they flip they become different numbers
- Return "changed"

Problem – 788. Rotated Digits

Medium

LeetCode leetcode.com/problems/rotated-digits

Code

Time: **O(n log n)** Space: **O(1)**

For each number, we examine each of its digits:

- A number i has $\log_{10}(i)$ digits → in worst case: O(log n) per number

```
int rotatedDigits(int n) {
    int count = 0;
    for (int i = 1; i <= n; ++i) {
        if (isGood(i)) count++;
    }
    return count;
}

bool isGood(int num) {
    bool changed = false;
    while (num > 0) {
        int digit = num % 10;
        if (digit == 3 || digit == 4 || digit == 7) return false;
        if (digit == 2 || digit == 5 || digit == 6 || digit == 9)
changed = true;
        num /= 10;
    }
    return changed;
}
```

BINARY

Bit Manipulation in C

- **Operators**

  & AND **|** OR **^** XOR **~** NOT **<<** LEFT SHIFT **>>** RIGHT SHIFT

- **Common Operations**

  set bit: `num |= (1 << pos)`

  clear bit: `num &= ~(1 << pos)`

  toggle bit: `num ^= (1 << pos)`

  check bit: `(num & (1 << pos)) != 0`

  extract bit: `(num >> pos) & 1`

  extract a range of bits: `(num >> pos) & ((1 << length) - 1)`

- **Example**

```
void copyBit(int *dst, int src, int srcPos, int dstPos) {
      int bit = (src >> srcPos) & 1;  // extract bit
      *dst &= ~(1 << dstPos);  // clear destination bit
      *dst |= (bit << dstPos);  // set destination bit
}
```

Binary

- In C++, **`std::bitset`** represents a fixed-size sequence of N bits
- Example:

```
std::bitset<8> bitmask;
bitmask.reset(1)
bitmask.set(1)
if (bitmask.test(1)) { // true
…
```

- **reset** : set bit to false
- **set** : set a specific bit
- **test** : check a specific bit
- **count** : return the number of bits set to true
- **flip** : toggle the value of the bits (if true, set to false and vice-versa)

Problem – 371. Sum of Two Integers

Medium

LeetCode leetcode.com/problems/sum-of-two-integers

Problem

- …

Problem – 371. Sum of Two Integers

Medium

LeetCode leetcode.com/problems/sum-of-two-integers

Solution

- …

Problem – 371. Sum of Two Integers

Medium

LeetCode leetcode.com/problems/sum-of-two-integers

Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 191. Number of 1 Bits

Easy

LeetCode leetcode.com/problems/number-of-1-bits

Problem

- …

Problem – 191. Number of 1 Bits

Easy

LeetCode leetcode.com/problems/number-of-1-bits

Solution

- ...

Problem – 191. Number of 1 Bits

Easy

LeetCode leetcode.com/problems/number-of-1-bits

Code

- …

Problem – 191. Number of 1 Bits

Easy

LeetCode leetcode.com/problems/number-of-1-bits

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 338. Counting Bits

Easy

LeetCode leetcode.com/problems/counting-bits

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 338. Counting Bits

Easy

LeetCode leetcode.com/problems/counting-bits

Problem

- …

Problem – 338. Counting Bits

Easy

LeetCode leetcode.com/problems/counting-bits

Solution

- …

Problem – 338. Counting Bits

Easy

LeetCode leetcode.com/problems/counting-bits

Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 268. Missing Number

Easy

LeetCode https://leetcode.com/problems/missing-number

Problem

- …

Problem – 268. Missing Number

Easy

LeetCode https://leetcode.com/problems/missing-number

Solution

- …

Problem – 268. Missing Number

Easy

LeetCode https://leetcode.com/problems/missing-number

Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 190. Reverse Bits

Easy

LeetCode leetcode.com/problems/reverse-bits

Problem

- …

Problem – 190. Reverse Bits

Easy

LeetCode leetcode.com/problems/reverse-bits

Solution

- …

Problem – 190. Reverse Bits

Easy

LeetCode leetcode.com/problems/reverse-bits

Code

Time: **O(-)** Space: **O(-)**

- …

Negabinary

- Non-standard positional numeral system that uses base of -2
- Allow representing negative numbers in binary
- Example:

  1101_{-2}

  $(-2)^3 + (-2)^2 + 0 + (-2)^0 = -8 + 4 + 0 + 1 = -3$

Summing Negabinary

- Add as a regular binary number, but with **negative carry**

  $0 + 0 = 0$

  $1 + 0 = 1$

  $1 + 1 = 0$ with a negative carry 1

  $\mathbf{1} + 1 = 0$ (subtract)

  $\mathbf{1} + 0 = 1$ with a positive carry 1

Negabinary

Example 1

$$\begin{array}{r} 11\ \ 1 \\ 1011 \\ +\,1110 \\ \hline =\,\boxed{110001} \end{array}$$

1 + 0 = 1

1 + 1 = 0 with negative carry 1

1 + 1 = 0

1 + 1 = 0 with negative carry 1

1 + 0 = 1 with positive carry 1

1 + 0 = 1

red 1 = negative carry

green 1 = regular carry

Example 2

$$\begin{array}{r} 1111 \\ 101010 \\ +\,101100 \\ \hline =\,11110110 \end{array}$$

Reference

https://math.stackexchange.com/questions/3251605/how-to-add-negabinary-numbers

Problem 1073 – Adding Two Negabinary Numbers

Medium

https://leetcode.com/problems/adding-two-negabinary-numbers

Given two numbers **arr1** and **arr2** in base -2, return the result of adding them together.

Each number is given in *array format*: as an array of 0s and 1s, from most significant bit to least significant bit. For example, **arr = [1,1,0,1]** represents the number **(-2)^3 + (-2)^2 + (-2)^0 = -3.** A number **arr** in array, format is also guaranteed to have no leading zeros: either **arr == [0]** or **arr[0] == 1**.

Return the result of adding **arr1** and **arr2** in the same format: as an array of 0s and 1s with no leading zeros.

Example 1

```
Input: arr1 = [1,1,1,1,1], arr2 = [1,0,1]
Output: [1,0,0,0,0]
Explanation: arr1 represents 11, arr2 represents 5, the output represents 16.
```

Example 2

```
Input: arr1 = [0], arr2 = [0]
Output: [0]
```

Example 3

```
Input: arr1 = [0], arr2 = [1]
Output: [1]
```

Solution 1073 – Adding Two Negabinary Numbers

Medium

https://leetcode.com/problems/adding-two-negabinary-numbers

GRAPH (BFS/DFS)

Problem - Keys and Rooms

Medium

https://leetcode.com/problems/keys-and-rooms

```
int maxProfit(vector<int>& prices) {
    int profit = 0;
    int buy = prices[0];
    for (auto 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;
}
```

Problem – Clone Graph

Medium

LeetCode https://leetcode.com/problems/clone-graph

Problem Statement

- Given a node reference, create a deep copy of the graph
- The class node has two variables: val and neighbours

```
class Node {
  public int val;
  public List<Node> neighbors;
}
```

- **Output** is the node reference of the copy

Solution – Clone Graph

Medium

LeetCode https://leetcode.com/problems/clone-graph

Solution

- First check the edge cases (is the node null?)
- Create a hash map to store the nodes that is already created

```
unordered<int, Node*> graph;
```

- Check if the current node already exists in the graph
- If not, create a new Node object and store in the hashmap
- Visit all the neighbors and add the neighbors to this current node

Code – Clone Graph

Medium

LeetCode https://leetcode.com/problems/clone-graph

```
std::unordered_map<int, Node*> graph;

Node* cloneGraph(Node* node) {
    if (node == NULL) {
        return NULL;
    }
    // does this node object exists?
    if (graph.find(node->val) == graph.end()) {
        // node wasn't visited yet, store in the hashmap
        graph[node->val] = new Node(node->val);
        // visit all neighbours
        for (const auto& n : node->neighbors) {
            graph[node->val]->neighbors.push_back(cloneGraph(n));
        }
    }
    return graph[node->val];
}
```

Problem – 207. Course Schedule

Medium

LeetCode leetcode.com/problems/course-schedule

Problem

- You are given the number of courses and a course pre-requisite array
- Course pre–requisite indicates the dependency between courses
- **Example**:

  `numCourses = 2, prerequisites = [[1,0],[0,1]]`

  To take course 1, you must take course 0 first

  to take course 0, you must take course 1 first
- In this example, this schedule is not possible since one course depends on the other
- Return if the schedule is valid or not

Solution – 207. Course Schedule

Medium

LeetCode leetcode.com/problems/course-schedule

Solution

- Model as a graph problem
- Create a dependency graph between courses: **course A** depends on **course B**
- If there is a cycle, **A** to **B** and **B** to **A**, the schedule is invalid
- For the implementation: first convert the schedule to adjacency list
- Use DFS and track two status: **VISITING** and **VISITED**
- Go over each node in the adjacency list, and perform a DFS
- Once you find a node which status is **VISITING**, you've detected a cycle

Code – 207. Course Schedule

Medium

LeetCode leetcode.com/problems/course-schedule

Code

Time: **O(n + p)** Space: **O(n + p)** where n is the number of courses and p the number of edges in the graph

```cpp
enum class VisitState {
    NOT_VISITED,
    VISITING,
    VISITED
};

bool hasCycle(int node, const unordered_map<int, vector<int>>& adjList,
              unordered_map<int, VisitState>& visited) {
    // if we are revisiting a node in the current path, there's a cycle
    if (visited[node] == VisitState::VISITING) return true;

    // if we've already completed visiting this node, no need to check again
    if (visited[node] == VisitState::VISITED) return false;

    // mark the node as being visited
    visited[node] = VisitState::VISITING;

    for (int neighbor : adjList.at(node)) {
        if (hasCycle(neighbor, adjList, visited)) return true;
    }

    // mark the node as fully visited
    visited[node] = VisitState::VISITED;
    return false;
}
```

```cpp
bool canFinish(int numCourses, const vector<vector<int>>& prerequisites) {
    // build the adjacency list: course -> list of its prerequisites
    unordered_map<int, vector<int>> adjList;
    for (const auto& dependencyPair : prerequisites) {
        int course = dependencyPair[0];
        int prerequisite = dependencyPair[1];
        adjList[course].push_back(prerequisite);
    }

    unordered_map<int, VisitState> visited;

    // check each course for cycles
    for (int course = 0; course < numCourses; ++course) {
        if (adjList.count(course)) {
            if (hasCycle(course, adjList, visited)) return false;
        }
    }

    return true; // no cycles detected
}
```

Code – 207. Course Schedule

Medium

LeetCode leetcode.com/problems/course-schedule

Code (simplified)

Time: **O(n + p)** Space: **O(n + p)** where n is the number of courses and p the number of edges in the graph

```
bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
    // construct the graph
    vector<vector<int>> adjList(numCourses);
    // states:
    // unvisited = 0, visiting = -1, visited = 1
    vector<int> visited(numCourses, 0);
    for (const auto& pre : prerequisites) {
        adjList[pre[1]].push_back(pre[0]);
    }
    for (int n = 0; n < adjList.size(); ++n) {
        if (hasCycle(adjList, visited, n /* starting node */)) {
            return false;
        }
    }
    return true;
}

bool hasCycle(vector<vector<int>>& adjList, vector<int>& visited, int node) {
    if (visited[node] == -1) return true;
    if (visited[node] == 1) return false;

    // visiting
    visited[node] = -1;
    for (const auto& n: adjList[node]) {
        if (hasCycle(adjList, visited, n)) {
            return true;
        }
    }
    // already visited
    visited[node] = 1;
}
```

Problem – 417. Pacific Atlantic Water Flow

Medium

LeetCode leetcode.com/problems/pacific-atlantic-water-flow

Problem

- …

Solution – 417. Pacific Atlantic Water Flow

Medium

LeetCode leetcode.com/problems/pacific-atlantic-water-flow

Solution

- …

Code – 417. Pacific Atlantic Water Flow

Medium

LeetCode leetcode.com/problems/pacific-atlantic-water-flow

Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 200. Number of Islands

Medium

LeetCode leetcode.com/problems/number-of-islands

Problem

- …

Solution – 200. Number of Islands

Medium

LeetCode leetcode.com/problems/number-of-islands

Solution

- …

Code – 200. Number of Islands

Medium

LeetCode leetcode.com/problems/number-of-islands

Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 128. Longest Consecutive Sequence

Medium

LeetCode leetcode.com/problems/longest-consecutive-sequence

Problem

- …

Solution – 128. Longest Consecutive Sequence

Medium

LeetCode leetcode.com/problems/longest-consecutive-sequence

Solution

- …

Code – 128. Longest Consecutive Sequence

Medium

LeetCode leetcode.com/problems/longest-consecutive-sequence

Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 261. Graph Valid Tree

Medium

LeetCode leetcode.com/problems/graph-valid-tree

Problem

- …

Solution – 261. Graph Valid Tree

Medium

LeetCode leetcode.com/problems/graph-valid-tree

Solution

- ...

Code – 261. Graph Valid Tree

Medium

LeetCode leetcode.com/problems/graph-valid-tree

Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 323. Number of Connected Components

Medium

LeetCode leetcode.com/problems/number-of-connected-components-in-an-undirected-graph

Problem

- You are given a graph of **n** nodes, and an array of **edges (source, destination)**
- Edges indicates the edge between node **source** and **destination**
- Find the total number of isolated components (subgraphs)
- **Example:**

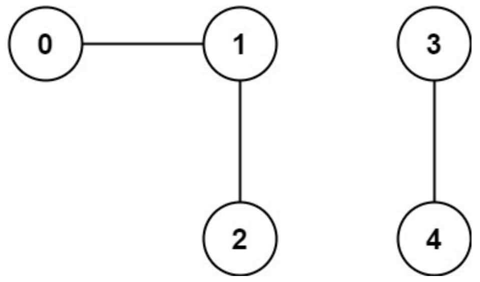
  Input:

  `n = 5, edges = [[0,1],[1,2],[3,4]]`

  Output: 2

  0 is connected to 1, 1 is connected to 2

  3 is a new subgraph connected to 4



Solution – 323. Number of Connected Components

Medium

LeetCode leetcode.com/problems/number-of-connected-components-in-an-undirected-graph

Solution

- Build an adjacency list from edges
- From each node, check if its visited
- If it is not visited, mark as a new “component” or subgraph
- Perform a DFS from that node

Code – 323. Number of Connected Components

Medium

LeetCode leetcode.com/problems/number-of-connected-components-in-an-undirected-graph

Code

Time: **O(n + E)** Space: **O(n + E)** where n is the number of nodes and E is edges size

```cpp
int countComponents(int n, vector<vector<int>>& edges) {
    vector<bool> visited(n, false);
    vector<vector<int>> adjList(n);

    // build adjancency list
    for (const auto& edge: edges) {
        adjList[edge[0]].push_back(edge[1]);
        adjList[edge[1]].push_back(edge[0]);
    }
    int totalComponents = 0;
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) {
            dfs(adjList, visited, i);
            totalComponents++;
        }
    }

    return totalComponents;
}

void dfs(vector<vector<int>>& adjList, vector<bool>& visited, int node) {
    if (visited[node]) return;
    visited[node] = true;
    for (const auto& neighbour : adjList[node]) {
        dfs(adjList, visited, neighbour);
    }
}
```

Problem – Maximum Level Sum of a Binary Tree

Medium

LeetCode https://leetcode.com/problems/maximum-level-sum-of-a-binary-tree

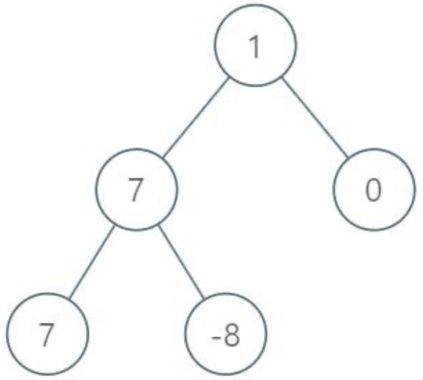
Problem Statement

- Given the root of a binary tree, find the smallest level with the maximum sum
- For example, the tree below has the follow sums for each level:

  level 1 (root) = 1

  level 2 = 7 + 0 = 7

  level 3 = 7 – 8 = -1
- Therefore, **level 2** has the maximum sum



Solution – Maximum Level Sum of a Binary Tree

Medium

LeetCode https://leetcode.com/problems/maximum-level-sum-of-a-binary-tree

Solution

- Have a queue with the nodes for the current level
- Sum the values from that level by taking the nodes from the queue
- Example, we know that level 1 has one node. Hence, pop the first node from the queue

  If level 2 has 2 nodes, pop two nodes, sum the values
- In addition, add left and right to the end of the queue to process the next level

Code – Maximum Level Sum of a Binary Tree

Medium

LeetCode https://leetcode.com/problems/maximum-level-sum-of-a-binary-tree

```cpp
int maxLevelSum(TreeNode* root) {
    std::queue<TreeNode*> nodes;
    int currentLevel = 0;
    int maxLevel = 1;
    int maxSum = INT_MIN;

    nodes.push(root);

    // traverse the graph
    while(!nodes.empty()) {
        int levelSum = 0;
        int levelSize = nodes.size();
        currentLevel++;

        // sum the values in current level
        for (int i = 0; i < levelSize; ++i) {
            TreeNode* node = nodes.front();
            levelSum += node->val;
            nodes.pop();

            if (node->left) nodes.push(node->left);
            if (node->right) nodes.push(node->right);
        }

        if (levelSum > maxSum) {
            maxLevel = currentLevel;
            maxSum = levelSum;
        }
    }

    return maxLevel;
}
```

Problem – 1236. Web Crawler

Medium

LeetCode https://leetcode.com/problems/web-crawler

Problem

- You are given a starting URL startURL and an interface HtmlParser with a method getUrls(url)
- getUrls(url) returns a vector of strings with the URLs found on the given page
- Start crawling from startUrl and recursively visit all reachable URLs
- Only visit URLs that share the same hostname as startUrl
- Return a list of all visited URLs (in any order)

Solution – 1236. Web Crawler

Medium

LeetCode https://leetcode.com/problems/web-crawler

Solution

- This is a graph problem framed as an object
- Both BFS and DFS are valid options
- Each url represent a node, and `getUrls` retrieve the neighbours
- Visit each node and add to the result if they have the same hostname

Code – 1236. Web Crawler

Medium

LeetCode https://leetcode.com/problems/web-crawler

Code (BFS)

Time: **O(n + m)** Space: **O(n + w)** where **n** is the number of unique URLs, **m** the number of links (edges), **w** is the explicit queue

```cpp
vector<string> crawl(string startUrl, HtmlParser htmlParser) {
    string hostname = getHostname(startUrl);
    queue<string> urls;
    unordered_set<string> visited;
    vector<string> result;

    urls.push(startUrl);
    visited.insert(startUrl);
    result.push_back(startUrl);

    while(!urls.empty()) {
        string url = urls.front();
        urls.pop();
        for (const auto& u : htmlParser.getUrls(url)) {
            // is it the same hostname?
            // have I already visited this one?
            if (visited.count(u)) continue;
            if (getHostname(u) != hostname) continue;
            urls.push(u);
            result.push_back(u);
            visited.insert(u);
        }
    }
    return result;
}
```

```cpp
string getHostname(const string& url) {
    int start = url.find("://") + 3;
    int end = url.find("/", start);
    return url.substr(start, end - start);
}
```

Problem – 1236. Web Crawler

Medium

LeetCode https://leetcode.com/problems/web-crawler

Code (DFS)

Time: **O(n + m)** Space: **O(n + h)** where **n** is the number of unique URLs, **m** the number of links (edges), **h** is the recursive stack

```
string getHostname(const string& url) {
    int start = url.find("://") + 3;
    int end = url.find("/", start);
    return url.substr(start, end - start);
}

void dfs(const string& hostname, const string& url, HtmlParser& htmlParser,
unordered_set<string>& visited, vector<string>& result) {
    if (visited.count(url)) return;
    result.push_back(url);
    visited.insert(url);

    for (const auto& u: htmlParser.getUrls(url)) {
        if (getHostname(u) == hostname) {
            dfs(hostname, u, htmlParser, visited, result);
        }
    }
}

vector<string> crawl(string startUrl, HtmlParser htmlParser) {
    string hostname = getHostname(startUrl);
    unordered_set<string> visited;
    vector<string> result;
    dfs(hostname, startUrl, htmlParser, visited, result);
    return result;
}
```

SHORTEST PATH

Shortest Path Algorithms

Algorithm	Use Case	Graph Type	Time Complexity	Notes
BFS	When all edges have **equal weight**	Unweighted / same-weighted edges	O(V + E)	Simplest and fastest when all weights are equal.
Dijkstra	When edge weights are **non-negative**	Weighted, no negative weights	O((V + E) log V) with heap	Greedy, efficient for SSSP (Single Source Shortest Path).
Bellman-Ford	When edge weights can be **negative**	Weighted, allows negative weights	O(V * E)	Slower, but handles negative weights and detects negative cycles.
Floyd-Warshall	For **all-pairs shortest paths**	Dense graphs, small number of nodes	O(V^3)	Easy to implement; handles negative weights (but not negative cycles).
A* Search	For shortest path with a **goal node** and **heuristic**	Weighted, heuristic needed	Depends on heuristic quality	Often used in pathfinding (e.g. maps, games); faster than Dijkstra if heuristic is good.
Johnson's Algorithm	**All-pairs shortest paths** in sparse graphs with **negative weights**	Weighted, allows negative weights	O(V^2 log V + V * E)	Reweights graph with Bellman-Ford, then runs Dijkstra from each node.
SPFA	Practical variant of Bellman-Ford, often faster	Weighted, allows negative weights	Avg: O(E), Worst: O(VE)	Queue-based; faster in practice, not guaranteed. Handles negative weights.
Bidirectional Search	When you know **start and target**, speeds up search in undirected graphs	Unweighted or uniformly weighted	O(b^(d/2)) in best case	Runs two simultaneous searches (from start and end); fast when goal is known.

Problem – 743. Network Delay Time

Medium

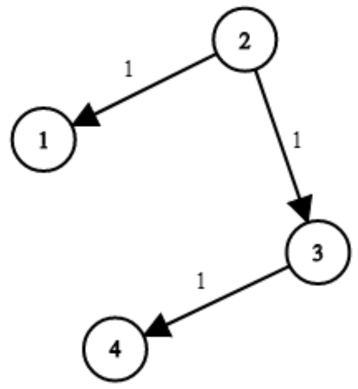
LeetCode leetcode.com/problems/network-delay-time

Problem

- You are given a network of nodes n with destination and time to reach that node
- You are given a starting node **k** and the number of nodes in the network **n**
- A signal is sent from node **k** to all nodes in the network
- Find the minimum time required for all nodes to receive the signal from **k**
- **Example:**

  k = 2 n =4

  Output: 3



Solution – 743. Network Delay Time

Medium

LeetCode leetcode.com/problems/network-delay-time

Solution

- This is solved using Djikstra algorithm
- Build the graph by storing in the destination node and the time from a source node:

  `graph[node] = [[node, distance]]`

  `graph[2] = [[1,1], [2,3]]`
- Set up a min-heap for Djikstra (priority_queue) with distance and node
- Perform Djikstra algorithm and store the shortest paths
- Check if all nodes were reached
- Return the longest distance among shortest paths

Code – 743. Network Delay Time

Medium

LeetCode leetcode.com/problems/network-delay-time

Code

Time: **O((n + e) log n)** Space: **O(n + e)** where n is the number of nodes and e the number of edges

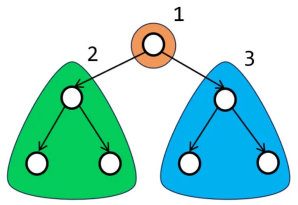
```cpp
int networkDelayTime(vector<vector<int>>& times, int n, int k) {
    // node => (destination, distance)
    unordered_map<int, vector<pair<int, int>>> graph;
    for (const auto& time : times) {
        // time[0] = source node, time[1] = dest node, time[2] = time
        graph[time[0]].emplace_back(time[1], time[2]);
    }

    // min heap: distance from the origin 'k' to 'node'
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> minHeap;
    minHeap.emplace(0, k); // distance, starting node

    // shortest path from each node to the origin ‘k’ (node, distance)
    unordered_map<int, int> dist;

    // we start exploring the nodes from the minimum
    // distance to the origin 'k'
    while (!minHeap.empty()) {
        auto [distance, node] = minHeap.top();
        minHeap.pop();
        // already visited, skip
        if (dist.count(node)) continue;
        // set the distance
        dist[node] = distance;
        // look at the connections
        for (const auto& [n, d] : graph[node]) {
            // n[node, distance]
            // quick optimization, not necessary
            if (dist.count(n)) continue;
            // add the distance since we want
            // the distance from the origin
            minHeap.emplace(distance + d, n);
        }
    }

    // check if all nodes were visited
    if (dist.size() != n) return -1;

    // all the minimum distances are calculated
    // find the max one since we want to reach
    // all nodes
    int minTime = 0;
    for (const auto& [n, d] : dist) {
        minTime = max(minTime, d);
    }

    return minTime;
}
```

TREE

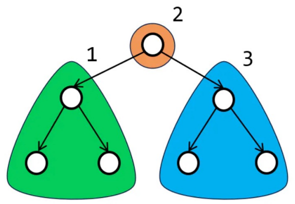
Tree Traversals

Depth-First Traversals

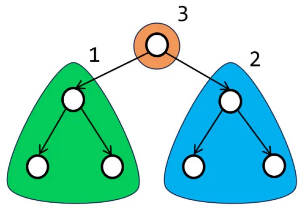
- **Pre-order**: Root – Left – Right



- **In-order**: Left – Root – Right



- **Post-order**: Left – Right – Root



Breadth-First Traversal (Level Order Traversal)

Visit every node on a level before moving to a lower level.

Image credits:
https://deriveit.org/coding/tree-dfs-113

Tree Traversals

Depth-First Traversals

Use a recursive algorithm to traverse according to the order

- **Pre-order**: Root – Left – Right ⇒

```
if (!root) return;
doSomething();
visit(node->left);
visit(node->right);
```

- **In-order**: Left – Root – Right ⇒

```
if (!root) return;
visit(node->left);
doSomething();
visit(node->right);
```

- **Post-order**: Left – Right – Root ⇒

```
if (!root) return;
visit(node->left);
visit(node->right);
doSomething();
```

Tree Traversals

Example of pre-order and in-order

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

// Pre-order traversal
void preorderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    cout << root->val << " ";
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

// In-order traversal
void inorderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    inorderTraversal(root->left);
    cout << root->val << " ";
    inorderTraversal(root->right);
}
```

Tree Traversals

Example of post-order and level-order

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

// Post-order traversal
void postorderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    cout << root->val << " ";
}

// Level-order traversal using a queue
void levelOrderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode* current = q.front();
        q.pop();
        cout << current->val << " ";
        if (current->left != nullptr) q.push(current->left);
        if (current->right != nullptr) q.push(current->right);
    }
}
```

BFS Using Stack

BFS with std::stack

- This might be useful for problems when you want to return and resume (for example, 872. Leaf-Similar Trees)

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

// Pre-order traversal
void bfs(std::stack<TreeNode*>& tree) {
    while(!tree.empty()) {
         TreeNode* root = tree.top();
         tree.pop();
         // do something ...
         if (root->right) tree.push(root->right);
         if (root->left) tree.push(root->left);
    }
}
```

Problem – 100. Same Tree

Easy

LeetCode leetcode.com/problems/same-tree

Problem

- You are given the root of two trees
- Write a function to check if they are the same
- **Example:**

  p = [1,2,3], q = [1,2,3]

  Output: true

Problem – 100. Same Tree

Easy

LeetCode leetcode.com/problems/same-tree

Solution

- Traverse both trees (**p** and **q**) recursively and check if the nodes are the same
- Start by the base case:

  are **p** and **q** null? return true
- One of them are null? return false, because they should be the same
- Finally, check if **p->val** is equal to **q->val** and also for both and left, recursively

Problem – 100. Same Tree

Easy

LeetCode leetcode.com/problems/same-tree

Code

Time: **O(n)** where n is the number of nodes Space: **O(h)** where h is the height of the tree. Best case is usually O(log n) for balanced trees, but skewed trees is usually O(n)

```
bool isSameTree(TreeNode* p, TreeNode* q) {
    // base case: leaf is null. If both are null, then return true
    if (!p && !q) return true;
    // if both are not NULL, then they must have value.
    // If one of them doesn't have value, then they're different, return false
    if (!p || !q) return false;
    // they must have the same value
    // as any other nodes in the tree
    return p->val == q->val &&
           isSameTree(p->left, q->left) &&
           isSameTree(p->right, q->right);
}
```

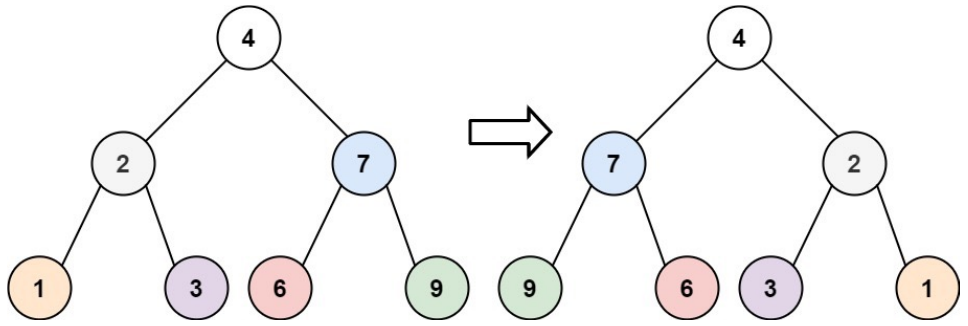
Problem – 226. Invert Binary Tree

Easy

LeetCode leetcode.com/problems/invert-binary-tree

Problem

- You are given the root of a binary tree
- Invert the tree and return the root
- **Example:**



Problem – 226. Invert Binary Tree

Easy

LeetCode leetcode.com/problems/invert-binary-tree

Solution

- Recursively traverse the tree
- Create a new pointer **temp** that points to **left** node
- Set **left** node to **right**
- Set **right** node to **temp**
- Call the function recursively for **left** and **right**
- Return root

Problem – 226. Invert Binary Tree

Easy

LeetCode leetcode.com/problems/invert-binary-tree

Code

Time: **O(n)** Space: **O(h)** where h is the height of the tree

```
TreeNode* invertTree(TreeNode* root) {
    // base case
    if (!root) return nullptr;

    // create a new pointer to left
    TreeNode* temp = root->left;
    // invert
    root->left = root->right;
    root->right = temp;

    // recursively invert left and right
    invertTree(root->left);
    invertTree(root->right);

    return root;
}
```

Problem – Maximum Depth of Binary Tree

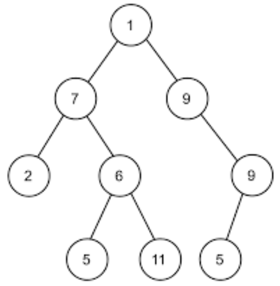
Easy

LeetCode https://leetcode.com/problems/maximum-depth-of-binary-tree

Problem Statement

- Given the root of a binary tree, find the *maximum depth*
- **Example:**

  root = [1,7,9,2,6,null,9,null,null,5,11,5,null]
- **Output**: 4



Solution – Maximum Depth of Binary Tree

Easy

LeetCode https://leetcode.com/problems/maximum-depth-of-binary-tree

Solution

- Perform **post-order** traversal: left - right - root
- Recursively go left and right to find each value
- Return the max of each one

Code – Maximum Depth of Binary Tree

Easy

LeetCode https://leetcode.com/problems/maximum-depth-of-binary-tree

```cpp
int maxDepth(TreeNode* root) {
    if (!root) return 0;
    // find max left
    int maxLeft = maxDepth(root->left);
    // find max right
    int maxRight = maxDepth(root->right);
    // return max +1 (account for root)
    return std::max(maxLeft, maxRight) + 1;
}
```

Problem – Path Sum

Easy

LeetCode https://leetcode.com/problems/path-sum

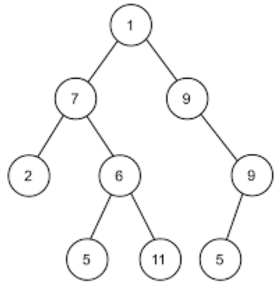
Problem Statement

- It is given the **root** of a binary tree and an integer **target sum**
- **Example:**

  root = [1,7,9,2,6,null,9,null,null,5,11,5,null]

  target sum = 10
- Return true if there is a path from root to leaf that adds up to 10
- **Output**: true

  Node 1 + Node 7 + Node 2 = 10



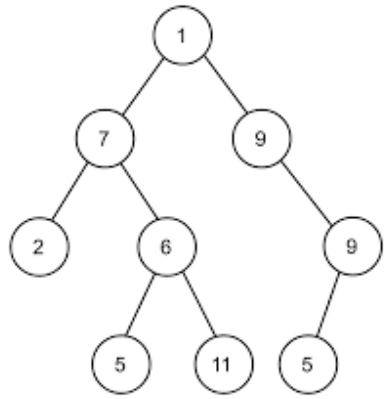
Solution – Path Sum

Easy

LeetCode https://leetcode.com/problems/path-sum

Solution

- Start from root node (1)
- Subtract from target number (example 10 – 1 = 9)
- Continue going down the tree, until the target is 0, return true
- After visiting all nodes, if the target is not zero, return false



Code – Path Sum

Easy

LeetCode https://leetcode.com/problems/path-sum

```
bool hasPathSum(TreeNode* root, int targetSum) {
    if (!root) {
        return false;
    }
    // we want targetSum to be zero
    targetSum -= root->val;
    // if there is no left, no right, we've reached the end of the path
    // so if the targetSum is zero, then the nodes summed up to the targetSum
    if (!root->left && !root->right && targetSum == 0) {
        return true;
    }
    // propagate to left and right
    return hasPathSum(root->left, targetSum) || hasPathSum(root->right, targetSum);
}
```

Also, a small performance tweak can be made by avoiding writing *targetSum:* *`targetSum -= root->val`*

This will avoid a memory write access, making the calculation directly in the CPU, but also at a cost of readability

```
    if (!root->left && !root->right && targetSum - root->val == 0) {
     …
    return hasPathSum(root->left, targetSum - root->val) || hasPathSum(root->right, targetSum - root->val);
```

Problem – 297. Serialize and Deserialize Binary Tree

Hard

LeetCode leetcode.com/problems/serialize-and-deserialize-binary-tree

Problem

- Design an algorithm to serialize and deserialize a binary tree
- You have to build two interfaces: serialize that returns a string, and deserialize that returns the whole tree as TreeNode pointer
- The string can be represented at any format (comma-separated, space separated etc)

Solution – 297. Serialize and Deserialize Binary Tree

Hard

LeetCode leetcode.com/problems/serialize-and-deserialize-binary-tree

Solution

- **Serialize**: traverse the tree pre-order, and append its value to a string

  Null value should also be represented

  Example: [1,2,null,null,3 ...]

  Call "traverse" to do it recursively
- **Deserialize**: split the string into tokens

  read each token and re-build the tree by adding a new node

  Call "buildTree" to do it recursively

Code – 297. Serialize and Deserialize Binary Tree

Hard

LeetCode leetcode.com/problems/serialize-and-deserialize-binary-tree

Code Time: O() Space: O()

```
string serialize(TreeNode* root) {
    // traverse the tree in pre-order: root, left, right
    // generate a string with comma separator,
    // example: 1,2,N,N,3 ...
    string result;
    traverse(root, result);
    return result;
}

TreeNode* deserialize(string data) {
    // split the input data
    vector<string> tokens = split(data);
    // index to be used to access the elements from tokens recursively.
    // Hence, we need to create it here to pass by reference.
    // Note that index is bounded by the number of tokens, so it won't overflow
    int index = 0;
    TreeNode* root = buildTree(tokens, index);
    return root;
}
```

continue...

Code – 297. Serialize and Deserialize Binary Tree

Hard

LeetCode leetcode.com/problems/serialize-and-deserialize-binary-tree

```cpp
TreeNode* buildTree(vector<string>& tokens, int& index) {
        // read the current token based on the index
        const string& token = tokens[index];
        // increment index before checking for null
        ++index;
        // base case: null node
        if (token == "N") {
            return nullptr;
        }
        // build root
        TreeNode* node = new TreeNode(stoi(token));
        // build left
        node->left = buildTree(tokens, index);
        // build right
        node->right = buildTree(tokens, index);
        return node;
    }
```

```cpp
// traverse in pre-order (root, left, right)
// and append the values to the string 's'
// append 'N' if it is NULL
void traverse(TreeNode* root, string& s) {
    if (!s.empty()) s += ",";
    // base case, we need to append null
    if (!root) {
        s += "N";
        return;
    }
    // visit root
    s += to_string(root->val);
    // visit left
    traverse(root->left, s);
    // visit right
    traverse(root->right, s);
}

// helper function in C++ to split string
vector<string> split(const string& s) {
    vector<string> result;
    stringstream ss(s);
    string token;
    while(getline(ss, token, ',')) {
        result.push_back(token);
    }
    return result;
}
```

Code – 297. Serialize and Deserialize Binary Tree

Hard

LeetCode leetcode.com/problems/serialize-and-deserialize-binary-tree

Some interesting alternative to split

- C++ 23 have an interesting way to split using std::views::split

```
vector<string> split(string s) {
    auto result = s |
        views::split(',') |
        views::transform([](auto&& subRange) {
                return string(subRange.start(), subRange.end());
                });
}
```

- To understand, this follow a structure similar to unix pipes:

  echo "123,N,556" | split | transform

- std::views::split returns ranges, something like:

  [range("123"), range("N"), range("556")]

- std::views::transform converts each subrange into an actual string

Problem – Kth Smallest Element in a BST

Medium

LeetCode leetcode.com/problems/kth-smallest-element-in-a-bst

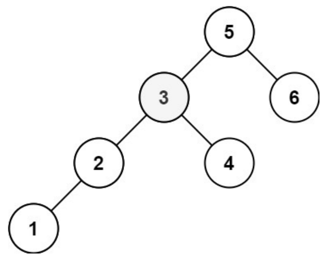
Problem Statement / Solution

- You are given the **root** of a binary search tree and an **integer k**
- Find the kth smallest value
- **Example**

  From all values in the tree: 1,2,3,4,5,6

  k = 3 so find the 3th smallest value

  Output is 3: 1,2,**3**,4,5,6 (3th)



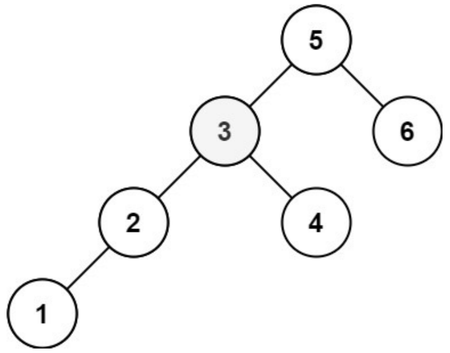
Solution – Kth Smallest Element in a BST

Medium

LeetCode leetcode.com/problems/kth-smallest-element-in-a-bst

Solution

- Note that the smallest element is in the left leaf
- Therefore, there is an order from small → big values from left → root → right
- Perform in-order traversal **k** times and stop in the desired node



Code – Kth Smallest Element in a BST

Medium

LeetCode leetcode.com/problems/kth-smallest-element-in-a-bst

Code

Time: **O(k)** Space: **O(h)** where **h** is the height of the tree

```
// in-order traversal: left, node: right
void traverse(TreeNode* node, int& k, int& result) {
    // base case
    if (!node) return;
    // visit left first
    traverse(node->left, k, result);
    // visit node
    k--;
    if (k == 0) {
        result = node->val;
        return;
    }
    // visit right
    traverse(node->right, k, result);
}

int kthSmallest(TreeNode* root, int k) {
    // perform pre-order traversal
    int result;
    traverse(root, k, result);
    return result;
}
```

Problem – 124. Binary Tree Maximum Path Sum

Hard

LeetCode leetcode.com/problems/binary-tree-maximum-path-sum

Problem

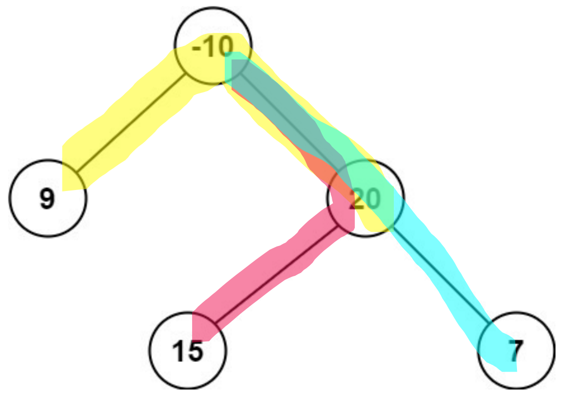
- You are given the root node of a binary tree
- Return the **max path sum** of any path
- A path can be linear (from the root all the way down to the leaf) or the three node: root, left and right)
- A path can start at any node
- **Example:**

  9 → -10 → 20 is a valid path

  9 → -10 → 20 → 7 is **NOT** a valid path

  -10 → 20 → 15 is a valid path

  20 → 7 is a valid path



Solution – 124. Binary Tree Maximum Path Sum

Hard

LeetCode leetcode.com/problems/binary-tree-maximum-path-sum

Solution

- Use **post-order** traversal (bottom-up recursion)
- At each node:

  - Recursively compute left and right max path gains

  - Consider all 3 possible paths:

    1. Turn path: left + root + right

    2. Linear path: root + left

    3. Linear path: root + right

  - Also consider just the root (if the children is negative)
- Track the maximum path seen so far
- Only return linear path (root + one child) upward to maintain the valid structure
- Also, prune negative gain before returning

Problem – 124. Binary Tree Maximum Path Sum

Hard

LeetCode leetcode.com/problems/binary-tree-maximum-path-sum

Code

Time: **O(n)** Space: **O(h)** where **n** is the number of nodes and **h** is the height of the tree.

```cpp
int findMaxSum(TreeNode* root, int& maxSum) {
    if (!root) return 0;
    int left = findMaxSum(root->left, maxSum);
    int right = findMaxSum(root->right, maxSum);
    // 1st possible path: exactly the only 3 nodes: root, right and left
    int threeNodes = left + right + root->val;
    // 2nd possible path, linear recursive path: root + left
    int secondPath = root->val + left;
    // 3rd possible path, linear recurrsive path: root + right
    int thirdPath = root->val + right;

    // check if we should consider left, right or only root itself
    int bestPath = max({root->val, secondPath, thirdPath});

    // maxSum can be the accumulated 2nd and 3rd (linear path)
    // or the threeNodes path
    maxSum = max({maxSum, bestPath, threeNodes});

    // Prune subtree: we start from the bottom, so we can set 0
    // to ignore left or right path
    return max(0, bestPath);
}

int maxPathSum(TreeNode* root) {
    int maxSum = INT_MIN;
    findMaxSum(root, maxSum);
    return maxSum;
}
```

Problem – 124. Binary Tree Maximum Path Sum

Hard

LeetCode leetcode.com/problems/binary-tree-maximum-path-sum

Code (compact)

Time: **O(n)** Space: **O(h)** where **n** is the number of nodes and **h** is the height of the tree.

```
int find(TreeNode *node, int& totalMax) {
    if (!node) return 0;
    int leftGain = max(0, find(node->left, totalMax));
    int rightGain = max(0, find(node->right, totalMax));
    int currentMax = node->val + leftGain + rightGain;
    totalMax = max(totalMax, currentMax);
    return node->val + max(leftGain, rightGain);
}

int maxPathSum(TreeNode* root) {
    int totalMax = INT_MIN;
    find(root, totalMax);
    return totalMax;
}
```

Problem – 102. Binary Tree Level Order Traversal

Medium

LeetCode leetcode.com/problems/binary-tree-level-order-traversal

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 572. Subtree of Another Tree

Easy

LeetCode leetcode.com/problems/subtree-of-another-tree

Problem

- You are given the root of a binary tree root and the root of another binary tree subRoot
- Determine whether subRoot is a **subtree** of root.
- A subtree of a binary tree is a node in the tree along with all of its descendants
- The tree itself is also considered a subtree.

Problem – 572. Subtree of Another Tree

Easy

LeetCode leetcode.com/problems/subtree-of-another-tree

Solution

- Define a helper function isSameTree(a, b) that checks if two trees rooted at a and b are identical in both structure and node values.
- Traverse the root tree, and for each node:
- Use isSameTree(node, subRoot) to check if a matching subtree starts at that node.
- Return true if any such match is found; otherwise, return false.

Problem – 572. Subtree of Another Tree

Easy

LeetCode leetcode.com/problems/subtree-of-another-tree

Code

Time: **O(m * n)** Space: **O(h)** where n is the number of nodes of the tree and m the number of nodes of the subtree, and h the height of the tree.

```
bool isSame(TreeNode* q, TreeNode* r) {
    // both are null, so they're the same
    if (!q && !r) return true;
    // if they're not null, both must be not null
    if (!q || !r) return false;
    // now check the values
    if (q->val != r->val) return false;
    // check left and right
    return isSame(q->left, r->left) && isSame(q->right, r->right);
}

bool isSubtree(TreeNode* root, TreeNode* subRoot) {
    if (!root) return false;
    // Check starting from the root first
    if (isSame(root, subRoot)) {
        return true;
    }
    // they are not the same starting from the root,
    // but still subRoot may be in the middle of root. So check it recursively
    return isSubtree(root->left, subRoot) || isSubtree(root->right, subRoot);
}
```

Problem – 872. Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Problem Statement

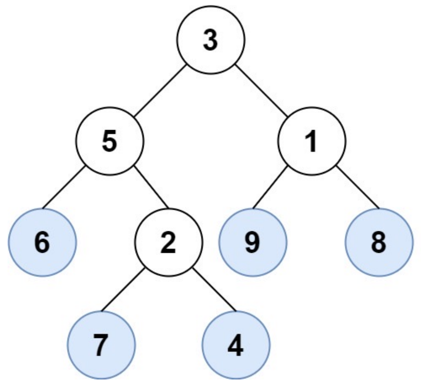
- You are given two trees
- The goal is to compare if they have the same leaves
- The leaves should be in the same order
- Example:

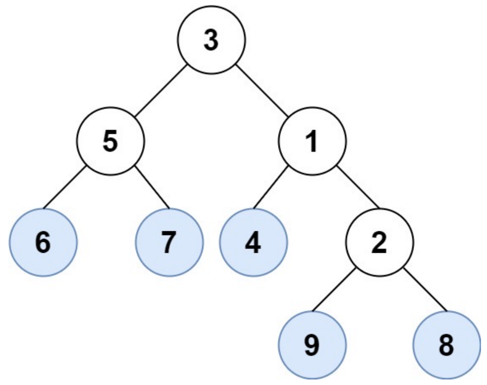
  First tree:

  leaves = 6,7,4,9,8 (blue nodes)

  Second tree:

  leaves = 6,7,4,9,8
- Return true if the leaves are the same





Solution – Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Solution

- Get the first leaf value from tree 1
- Get the first leaf value from tree 2
- Compare, if they are different, return false immediately
- Otherwise, continue finding the next leaf value for tree 1 and 2

Implementation

- Create two stacks **stack<TreeNode*> left** and **stack<TreeNode*> right**
- Add the

Code – Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Code

Time: **O(n + m)** where n and m are the numbers of nodes for trees 1 and 2   Space: **O(h1 + h2)** where h1 and h2 represents the height of the tree

```cpp
// returns the value of the leaf, or -1 if empty
int getLeaf(stack<TreeNode*>& tree) {
    // tree is a reference, we will always pop an element from it
    while(!tree.empty()) {
        // get the top element from the stack
        TreeNode* node = tree.top();
        // already visited, so remove from stack
        tree.pop();
        // is this a leaf?
        if (!node->left && !node->right) {
            // yes, return the value
            return node->val;
        }
        // push the right FIRST to the stack
        if (node->right) tree.push(node->right);
        // left should be on top of the stack
        if (node->left) tree.push(node->left);
    }
    return -1;
}
```

```cpp
bool leafSimilar(TreeNode* root1, TreeNode* root2) {
    // initialize the stacks, add root1 and root2
    std::stack<TreeNode*> leftTree, rightTree;
    leftTree.push(root1);
    rightTree.push(root2);

    while(true) {
        // get the leaves to compare
        int leaf1 = getLeaf(leftTree);
        int leaf2 = getLeaf(rightTree);
        // exit immediately if one leaf is different
        if (leaf1 != leaf2) return false;
        // stop when there are no leaves left
        if (leaf1 == -1 || leaf2 == -1) break;
    }
    return true;
}
```

Code – Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Code (another approach)

Time: **O(n + m)** where n and m are the numbers of nodes for trees 1 and 2   Space: **O(h1 + h2)** where h1 and h2 represents the height of the tree

```cpp
void extractLeafs(TreeNode* node, vector<int>& leafValues) {
    // base case, return
    if (!node) return;
    // if it looks like a leaf, no left child
    // like a leaf, no right child like a leaf,
    // then it's probably  a leaf
    // add to the vector
    if (!node->left && !node->right) {
        leafValues.push_back(node->val);
    }
    // continue looking at left and right
    extractLeafs(node->left, leafValues);
    extractLeafs(node->right, leafValues);
}

bool leafSimilar(TreeNode* root1, TreeNode* root2) {
    vector<int> tree1Values;
    vector<int> tree2Values;
    // extract all leafs from tree 1
    extractLeafs(root1, tree1Values);
    // extract all leafs from tree 2
    extractLeafs(root2, tree2Values);
    // compare
    return tree1Values == tree2Values;
}
```

Problem – 1448. Count Good Nodes in Binary Tree

Medium

LeetCode leetcode.com/problems/count-good-nodes-in-binary-tree

Problem Statement

- You are given a binary tree and have to find “**good**” **nodes**
- A **good node** is a **node** where the values in the path are always than the **node**
- The **root node** is always a **good node**
- Example:

  - root 3 is a good node

  left side:

  - left **leaf 1** is not a good node because **1 < 3**

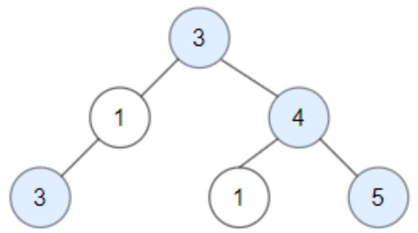
  - **leaf 3** is a good node because **3 > 1** and **3 == 3**

  right side:

  - **leaf 4** is a good node because **4 > 3**

  - **leaf 1** is not a good node because **1 < 4**

  - **leaf 5** is a good node because **5 > 4 > 3**



Solution – 1448. Count Good Nodes in Binary Tree

Medium

LeetCode leetcode.com/problems/count-good-nodes-in-binary-tree

Solution

- Use DFS traversal to explore the tree from the root to all leaf nodes
- As you traverse, **keep track of the maximum value** along the path from root to node
- Update max value once you find a node value greater than the max value
- **Recursive logic**

  Base case: if the node is *`nullptr`*, return 0

  At each node:

  - Compare its value to max so far
  - If it is a good node, increase a local count
  - Recursively repeat this process for the left and right children, passing along the updated max value

Code – 1448. Count Good Nodes in Binary Tree

Medium

LeetCode leetcode.com/problems/count-good-nodes-in-binary-tree

Code

Time: **O(n)** Space: **O(h)**

```
int traverse(TreeNode* root, int maxValue) {
    if (!root) return 0;
    // is this a good node?
    int count = 0;
    if (root->val >= maxValue) {
        maxValue = root->val;
        count = 1;
    }
    count += traverse(root->left, maxValue);
    count += traverse(root->right, maxValue);
    return count;
}

int goodNodes(TreeNode* root) {
    if (!root) return 0;
    return traverse(root, root->val);
}
```

INTERVAL

greedy strategy: sort by the end time

Because ending earlier gives **more room** for future intervals. It’s a classic greedy trick: choose the interval that **frees up time** as quickly as possible.

Problem – 57. Insert Interval

Medium

LeetCode leetcode.com/problems/insert-interval

Problem Statement

- You are given an array of **intervals,** where **intervals[i] = [start$_i$, end$_i$]** and **newInterval = [start, end]**
- **newInterval** must be inserted into **intervals**
- Overlapping intervals must be merged
- Example

  intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]] **newInterval** = [4,8]

  Output: [[1,2],[3,10],[12,16]]

Solution – 57. Insert Interval

Medium

LeetCode leetcode.com/problems/insert-interval

Solution

- Sort intervals by the first element (start)
- Initialize **result**
- Solve in three loops:

  1. While there is no overlap with **newInterval**, add to **intervals[i]** to **result**
  2. While it overlaps, merge **newInterval**
  3. While until the end intervals and add the remaining **intervals[i]**

Code – 57. Insert Interval

Medium

LeetCode leetcode.com/problems/insert-interval

Code

Time: **O(n)** Space: **O(n)** where n is the size of intervals

```cpp
vector<vector<int>> insert(vector<vector<int>>& intervals, vector<int>& newInterval) {
    vector<vector<int>> result;
    int tupleIndex = 0;
    int totalTuples = intervals.size();
    // 1. check if it overlaps
    // 1 ------ 2
    //               4 ----- 8
    while (tupleIndex < totalTuples && intervals[tupleIndex][1] < newInterval[0]) {
        result.push_back(intervals[tupleIndex]);
        ++tupleIndex;
    }


    // 2. merge overlap. We already know there is an overlap here,
    // otherwise it should be sorted out in the previous step
    // 3 ---- 5
    //     4 ----- 8
    while (tupleIndex < totalTuples && intervals[tupleIndex][0] <= newInterval[1]) {
        newInterval[0] = min(newInterval[0], intervals[tupleIndex][0]);
        newInterval[1] = max(newInterval[1], intervals[tupleIndex][1]);
        ++tupleIndex;
    }
    result.push_back(newInterval);


    // 3. add remaining parts
    while (tupleIndex < totalTuples) {
        result.push_back(intervals[tupleIndex]);
        ++tupleIndex;
    }
    return result;
}
```

Problem – 56. Merge Intervals

Medium

LeetCode leetcode.com/problems/merge-intervals

Problem Statement

- ...

Solution – 56. Merge Intervals

Medium

LeetCode leetcode.com/problems/merge-intervals

Solution

- …

Code – 56. Merge Intervals

Medium

LeetCode leetcode.com/problems/merge-intervals

Code

Time: **O(n)** Space: **O(n)**

- …

Problem – 435. Non-overlapping Intervals

Medium

LeetCode leetcode.com/problems/non-overlapping-intervals

Problem Statement

- …

Solution – 435. Non-overlapping Intervals

Medium

LeetCode leetcode.com/problems/non-overlapping-intervals

Solution

- …

Code – 435. Non-overlapping Intervals

Medium

LeetCode leetcode.com/problems/non-overlapping-intervals

Code

Time: **O(n)** Space: **O(n)**

- …

LINKED LIST

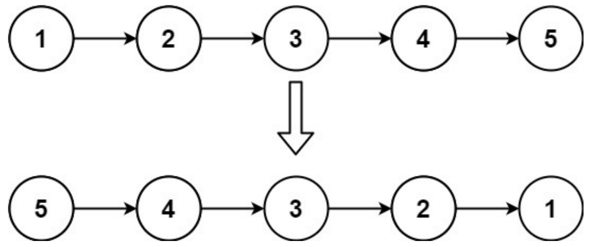
Problem – 206. Reverse Linked List

Easy

LeetCode leetcode.com/problems/reverse-linked-list

Problem

- This is a classic problem
- Given a singly linked list, reverse its order


$$\begin{tikzcd}1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] \arrow[d, Rightarrow] & 4 \arrow[r] & 5 \\ 5 \arrow[r] & 4 \arrow[r] & 3 \arrow[r] & 2 \arrow[r] & 1\end{tikzcd}$$

Solution – 206. Reverse Linked List

Easy

LeetCode leetcode.com/problems/reverse-linked-list

Solution

- Use recursive approach
- Looking at the pseudo-code, this recursion will return the last node:

```
reverseList(head) {
    if (!head->next) return head
    node = reverseList(head->next);
    return node
}
```

- From end to beginning, each head will be a node in the list
- Therefore, you can change this node by setting a new head:

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

Code – 206. Reverse Linked List

Easy

LeetCode leetcode.com/problems/reverse-linked-list

Code

Time: **O(n)** Space: **O(1)**

```
ListNode* reverseList(ListNode* head) {
    if (!head->next) return head;
    ListNode* node = reverseList(head->next);
    head->next->next = head;
    head->next = nullptr;
    return node;
}
```

Problem – 141. Linked List Cycle

Easy

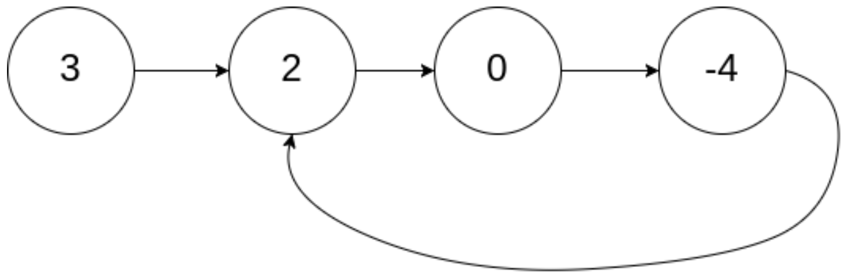
LeetCode leetcode.com/problems/linked-list-cycle

Problem

- You are given the head of a linked list
- Return **true** if there is a cycle, false otherwise
- **Example:**

  In the image below, there is a cycle (-4 to 2)

  Output: true



Solution – 141. Linked List Cycle

Easy

LeetCode leetcode.com/problems/linked-list-cycle

Solution

- Have two pointers: fast and slow
- Slow will go over each item in the linked list
- Fast will go twice as fast as slow (`fast = fast->next->next`)
- If fast reach at the end, there is no cycle
- If fast encounter slow, there is a cycle, return true

Code – 141. Linked List Cycle

Easy

LeetCode leetcode.com/problems/linked-list-cycle

Code

Time: **O(n)** Space: **O(1)**

```
bool hasCycle(ListNode *head) {
    if (!head || !head->next) return false;
    ListNode* slow = head;
    ListNode* fast = head;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) return true;
    }
    return false;
}
```

Problem – 21. Merge Two Sorted Lists

Easy

LeetCode leetcode.com/problems/merge-two-sorted-lists

Problem Statement / Solution / Code

Time: **O(n)** Space: **O(n)**

- …

Problem – 23. Merge k Sorted Lists

Hard

LeetCode leetcode.com/problems/merge-k-sorted-lists

Problem Statement / Solution / Code

Time: **O(n)** Space: **O(n)**

- …

Problem – 19. Remove Nth Node From End of List

Medium

LeetCode leetcode.com/problems/remove-nth-node-from-end-of-list

Problem Statement / Solution / Code

Time: **O(n)** Space: **O(n)**

- ...

Problem – 143. Reorder List

Medium

LeetCode leetcode.com/problems/reorder-list

Problem Statement / Solution / Code

Time: **O(n)** Space: **O(n)**

- …

Problem – 24. Swap Nodes in Pair

Medium

https://leetcode.com/problems/swap-nodes-in-pairs

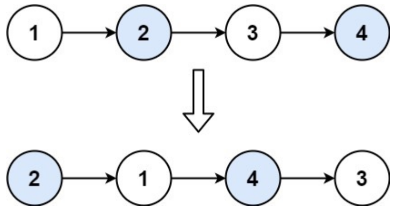
Problem

Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

Example 1

Input: head = [1,2,3,4]

Output: [2,1,4,3]



Example 2

Input: head = []

Output: []

Example 3:

Example 3

Input: head = [1]

Output: [1]

Solution – Swap Nodes in Pair

Medium

https://leetcode.com/problems/swap-nodes-in-pairs

```
ListNode* swapPairs(ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }
    ListNode *node = head;
    ListNode *prev = NULL;
    head = head->next;

    while (node && node->next) {
        ListNode *second = node->next;
        ListNode *next_pair = second->next;
        second->next = node;
        node->next = next_pair;
        if (prev) {
            prev->next = second;
        }
        prev = node;
        node = next_pair;
    }
    return head;
}
```

Solution (recursive) – Swap Nodes in Pair

Medium

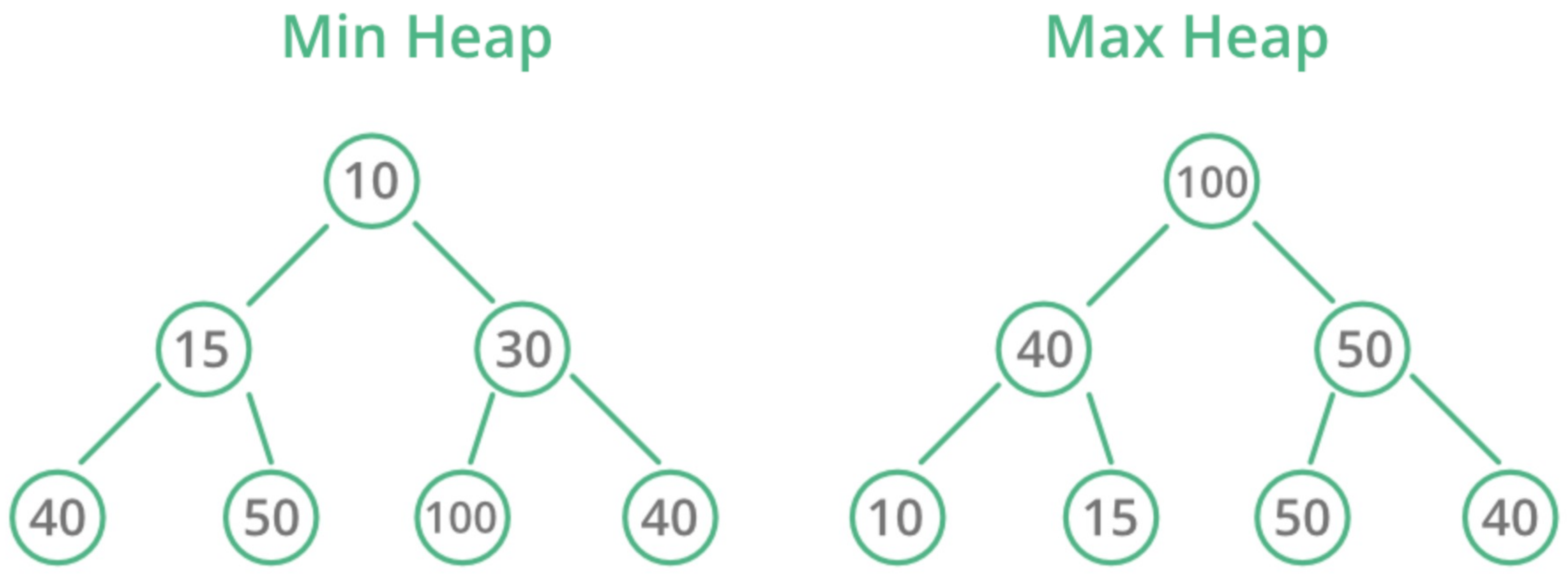
https://leetcode.com/problems/swap-nodes-in-pairs

```
ListNode* swapPairs(ListNode* head) {
    if(!head || !head->next)
        return head;
    ListNode* newHead = head->next;
    head->next = swapPairs(head->next->next);
    newHead->next = head;
    return newHead;
}
```

HEAP / PRIORITY QUEUE

Heap

- **Heap** is a complete binary tree that satisfy the heap property (max or min)
- **Min heap**: root node contains the minimum value
- **Max heap**: root node contains the maximum value



Heap in C++

Two main ways to implement:

1. Using **std::make_heap** from **<algorithm>**

```
std::make_heap(RandomIt first, RandomIt last)
std::push_heap(RandomIt first, RandomIt last)
std::pop_heap(RandomIt first, RandomIt last)
std::sort_heap(RandomIt first, RandomIt last)
```

2. Using **std::priority_queue** from **<queue>** **(recommended)**

```
std::priority_queue<T, Container, Compare>
```

Heap in C++ – std::priority_queue example

Min heap

`std::priority_queue<int, std::vector<int>, std::greater<int>>`

Max heap

`std::priority_queue<int>` or

`std::priority_queue<int, std::vector<int> std::less<int>>`

```
// Min heap
std::priority_queue<int, std::vector<int>, std::greater<int>> minHeap;

minHeap.push(3);
minHeap.push(6);
minHeap.push(4);
// remove top element (3)
minHeap.pop();
// root node (top) is now 4
std::cout << minHeap.top();
```

Problem – Kth Largest Element in an Array

Medium

https://leetcode.com/problems/kth-largest-element-in-an-array

Problem

Given an integer array **nums** and an integer **k**, return the **k**th largest element in the array.

Note that it is the **k**th largest element in the sorted order, not the **k**th distinct element.

Example 1

```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```

Example 2

```
Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4
```

Although this problem is classified as “medium”, in my opinion it should be classified as “easy”

Solution 1 – Kth Largest Element in an Array

Medium

https://leetcode.com/problems/kth-largest-element-in-an-array

```
// SOLUTION 1

int findKthLargest(vector<int>& nums, int k) {
    std::priority_queue<int, std::vector<int>, std::greater<int>> minHeap;
    for (const auto& num : nums) {
        if (minHeap.size() < k) {
            minHeap.push(num);
        } else if (num > minHeap.top()) {
            minHeap.pop();
            minHeap.push(num);
        }
    }
    return minHeap.top();
}
```

Solution 2 – Kth Largest Element in an Array

Medium

https://leetcode.com/problems/kth-largest-element-in-an-array

```
// SOLUTION 2 - Simpler approach

int findKthLargest(vector<int>& nums, int k) {
    // min heap: minimum values will be always at the top
    std::priority_queue<int, std::vector<int>, std::greater<int>> minHeap;
    for (const auto& num : nums) {
        // push each num to the heap
        minHeap.push(num);
        // we need the kth largest element only, so once after pushing more than k
        // elements, remove the smallest one (the top)
        if (minHeap.size() > k) {
            minHeap.pop();
        }
    }
    return minHeap.top();
}
```

Problem – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Problem

- You are given an **array** of numbers and an **integer** k
- Return an array with the **k** most frequent elements

Example

Input:

```
nums = [1,1,1,2,2,3], k = 2
```

Output:

```
[1,2]
```

Solution – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Solution (1) - hashmap + array sort

- Go over the array, count the numbers and store them in an *unordered_map*

  Example:

  ```
  nums = [1,1,1,2,2,3], k = 2
  freq[1] = 3
  freq[2] = 2
  ...
  ```

- Go over the *unordered_map*, add to an array and sort descending
- Create another array adding the **k** first elements and return

Code – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Code (1) Time: O(n log n) Space: O(n)

```
vector<int> topKFrequent(vector<int>& nums, int k) {
    // 1. Create the number's frequency map
    // O(n)
    unordered_map<int, int> freq;
    for (const auto& num : nums) {
        freq[num] += 1;
    }
    // 2. Create an array with the frequencies
    vector<pair<int, int>> freqVec(freq.begin(), freq.end());
    // 3. Sort by the frequency O(n log n)
    sort(freqVec.begin(), freqVec.end(), [](auto& a, auto& b) {
            return a.second > b.second;
            });
    // 4. Create the result with the k first elements
    // O(k)
    vector<int> result;
    for (int i = 0; i < k; ++i) {
        result.push_back(freqVec[i].first);
    }
    return result;
}
```

Solution – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Solution (2) - hashmap + min heap

- Go over the array, count the numbers and store them in an *unordered_map*

  Example:

  ```
  nums = [1,1,1,2,2,3], k = 2
  freq[1] = 3
  freq[2] = 2
  ...
  ```

- Go over the frequencies, add to a min heap. If the size of the heap exceeds **k**, remove the top one (the minimum value)
- Create another array result adding all elements from the heap and return it

Code – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Code (2)

Time: **O(n log k)** Space: **O(n)**

```cpp
vector<int> topKFrequent(vector<int>& nums, int k) {
    // 1. Create the number's frequency map
    // O(n)
    unordered_map<int, int> freq;
    for (const auto& num : nums) {
        freq[num] += 1;
    }
    // 2. Create the min heap with priority queue
    // O(n log k)
    priority_queue<pair<int, int>, vector<pair<int,int>>, greater<>> minHeap;
    for (const auto& [num, count] : freq) {
        minHeap.push({count, num});
        if (minHeap.size() > k) minHeap.pop();
    }
    // 3. build the result
    vector<int> result;
    while (!minHeap.empty()) {
        auto num = minHeap.top().second;
        minHeap.pop();
        result.push_back(num);
    }
    return result;
}
```

Solution – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Solution (3) - hashmap + bucket sort

- Go over the array, count the numbers and store them in an *unordered_map*

  Example:

  ```
  nums = [1,1,1,2,2,3], k = 2
  freq[1] = 3
  freq[2] = 2
  ...
  ```

- Create buckets for each frequency and add the corresponding numbers:

  `bucket[1] = [3]` → *3 only appears once in nums*

  `bucket[2] = [2]` → *2 appears twice*

  `bucket[3] = [1]` → *1 appears three times*

- Go over each bucket, add to the result and return it

Code – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Code (3)

Time: **O(n)** Space: **O(n)**

```
vector<int> topKFrequent(vector<int>& nums, int k) {
    // Create the number's frequency map
    unordered_map<int, int> freq;
    for (const auto& num : nums) {
        freq[num]++;
    }
    // create the buckets
    // e.g. [[1,2,3],[4,5,6]] ...
    vector<vector<int>> buckets(nums.size() + 1);
    for (const auto& [num, count] : freq) {
        buckets[count].push_back(num);
    }
    // go over each bucket to build the result
    vector<int> result;
    for (int i = buckets.size() - 1; i >= 0; --i) {
        for (const auto& num : buckets[i]) {
            result.push_back(num);
            if (result.size() == k) return result;
        }
    }
    return result;
}
```

Problem – 347. Top K Frequent Elements

Medium

LeetCode leetcode.com/problems/top-k-frequent-elements

Some considerations

- Theoretically, bucket sort should be the fastest solution **O(n) < O(n log k)**
- In practice, min heap end up being faster:

  - fewer allocations: priority_queue stores flat pairs rather than inner vectors

  - better cache locality: heap is built over a single array (binary heap)

  - if **k** is small, heap touches fewer elements

MATRIX

Problem – 73. Set Matrix Zeroes

Medium

LeetCode leetcode.com/problems/set-matrix-zeroes

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- ...

Problem – 54. Spiral Matrix

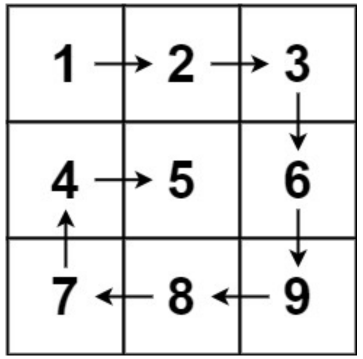
Medium

LeetCode leetcode.com/problems/spiral-matrix

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- …



Problem – 48. Rotate Image

Medium

LeetCode leetcode.com/problems/rotate-image

Problem Statement / Solution / Code

Time: O(-) Space: O(-)

- ...

DYNAMIC PROGRAMMING

Dynamic Programming

Dynamic Programming (DP) is an algorithm technique used to solve problems that can be broken down into **simpler, overlapping subproblems.**

Key Concepts of Dynamic Programming

- **Overlapping subproblems**: a problem has overlapping subproblems if it can be broken down into subproblems.
- **Memoization (Top-Down Approach)**: store the results in a cache (typically a dictionary or array) to avoid recalculation – recursion and caching approach.
- **Tabulation (Bottom-Up Approach)**: first solve all possible subproblems iteratively, and store them in a table.

Common Patterns in Dynamic Programming

- **Toy example (Fibonacci):** Climbing Stairs, N-th Tribonacci Number, Perfect Squares
- **Constant Transition:** Min Cost Climbing Stairs, House Robber, Decode Ways, Minimum Cost For Tickets, Solving Questions With Brainpower
- **Grid:** Unique Paths, Unique Paths II, Minimum Path Sum, Count Square Submatrices with All Ones, Maximal Square, Dungeon Game
- **Dual-Sequence:** Longest Common Subsequence, Uncrossed Lines, Minimum ASCII Delete Sum for Two Strings, Edit Distance, Distinct Subsequences, Shortest Common Supersequence
- **Interval:** Longest Palindromic Subsequence, Stone Game VII, Palindromic Substrings, Minimum Cost Tree From Leaf Values, Burst Balloons, Strange Printer
- **Longest Increasing Subsequence:** Count Number of Teams, Longest Increasing Subsequence, Partition Array for Maximum Sum, Largest Sum of Averages, Filling Bookcase Shelves
- **Knapsack:** Partition Equal Subset Sum, Number of Dice Rolls With Target Sum, Combination Sum IV, Ones and Zeroes, Coin Change, Coin Change II, Target Sum, Last Stone Weight II, Profitable Schemes
- **Topological Sort on Graphs:** Longest Increasing Path in a Matrix, Longest String Chain, Course Schedule III
- **DP on Trees:** House Robber III, Binary Tree Cameras
- **Other problems:** 2 Keys Keyboard, Word Break, Minimum Number of Removals to Make Mountain Array, Out of Boundary Paths

Credits
https://www.youtube.com/watch?v=9k31KcQmS_U
https://algo.monster/problems/dp-list

Dynamic Programming – Example – Fibonacci Sequence

Naive Recursive Approach $O(2^n)$

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

Memoization (Top-Down DP) $O(n)$

```
std::unordered_map<int, int> memo;

int fib(int n) {
    if (n <= 1) {
        return n;
    }
    if (memo.find(n) != memo.end()) {
        return memo[n];
    }
    memo[n] = fib(n - 1) + fib(n - 2);
    return memo[n];
}
```

Tabulation (Bottom-up DP) $O(n)$

```
int fib(int n) {
    if (n <= 1) {
        return n;
    }
    int dp[n + 1];
    dp[0] = 0;
    dp[1] = 1;
    for (int i = 2; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    return dp[n];
}
```

Problem – Climbing Stairs

Easy

LeetCode leetcode.com/problems/climbing-stairs

Problem Statement

You need to climb a staircase with n steps to get to the top. Each time you can choose to climb either **1 step** or **2 steps** at a time. Find out how many different ways you can climb to the top of the staircase.

Example 1

Input: n = 2

Output: 2

Explanation: There are two ways to get to the top

1. Climb 1 step at a time, twice
2. Climb 2 steps in one go

Example 2:

Input: n = 3

Output: 3

Explanation: There are three ways to get to the top:

1. Climb 1 step at a time, three times
2. Climb 1 step, then 2 steps
3. Climb 2 steps, then 1 ste.

Solution – Climbing Stairs

Easy

LeetCode leetcode.com/problems/climbing-stairs

```
std::unordered_map<int, int> memo;

int climbStairs(int n) {
    // Identify the sequence, when:
    // n = 0 (0 way), there is no way to get up
    // n = 1 (1 way): only one way : 1-step
    // n = 2 (2 ways): 1s + 1s | 2s
    // n = 3 (3 ways): 1s + 1s + 1s |  1s + 2s | 2s + 1s
    // n = 4 (5 ways): 1s + 1s + 1s + 1s | 1s + 1s + 2s | 1s + 2s + 1s | 2s + 1s + 1s | 2s + 2s |

    if (n <= 2) {
        return n;
    }

    if (memo.find(n) != memo.end()) {
        return memo[n];
    }

    memo[n] = climbStairs(n - 1) + climbStairs(n - 2);
    return memo[n];
}
```

Problem – 1143. Longest Common Subsequence

Medium

LeetCode https://leetcode.com/problems/longest-common-subsequence

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- ...

Problem – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Problem

- The robot is placed in a m x n grid
- It starts at the top-left cell (0,0) and must reach the bottom-right (m – 1, n – 1)
- The robot can only move right or down at any point
- Return the number of unique paths the robot can take to reach the destination

Start
Finish

Solution – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Solution 1 (recursive)

- Define a recursive function countPaths(m, n)
- **Base case**

  If m == 1 or n == 1, there’s only one way to reach that cell (either all downs or all rights).
- **Recursive case**

  To reach cell (m, n) the robot must come from:

  Cell (m - 1, n) → from above

  Cell (m, n - 1) → from left

  So the number of of paths to (m, n) is the **sum** of the paths to those two cells
- **Memoization**

  Use a 2D vector[m + 1][n + 1]

Code – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Code

Time: **O(m * n)** Space: **O(m * n)**

```
int countPaths(int m, int n, vector<vector<int>>& memo) {
    if (m == 1 || n == 1) return 1;
    if (memo[m][n] != -1) return memo[m][n];
    memo[m][n] = countPaths(m, n - 1, memo) + countPaths(m - 1, n,
memo);
    return memo[m][n];
}
int uniquePaths(int m, int n) {
    /*
      count(m, n) = count(m, n + 1) + count(m + 1, n)
      same as (imagine robot going from m,n to 0,0 up and left)
      count(m, n) = count(m, n - 1) + count(m - 1, n)
    */
    vector<vector<int>> memo(m + 1, vector<int>(n + 1, -1));
    return countPaths(m, n, memo);
}
```

Solution – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Solution 2 (iterative)

- Create a 2D vector dp of size (m+1) × (n+1) to store intermediate results
- Set dp[1][1] = 1 because there is exactly one way to stand on the starting cell
- Iterate through each cell (row, col) from (1, 1) to (m, n):

  - Skip (1, 1) since it's already initialized

  - For every other cell, the number of unique paths to it is the sum of:

    - Paths from the cell above: dp[row-1][col]

    - Paths from the cell to the left: dp[row][col-1]

    dp[row][col] = dp[row - 1][col] + dp[row][col - 1]

Code – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Code

Time: **O(m * n)** Space: **O(m * n)**

```cpp
int uniquePaths(int m, int n) {
    vector<vector<int>> dp(m + 1, vector<int>(n + 1));
    dp[1][1] = 1;
    for (int row = 1; row <= m; ++row) {
        for (int col = 1; col <= n; ++col) {
            if (row == 1 && col == 1) continue;
            dp[row][col] = dp[row-1][col] + dp[row][col-1];
        }
    }
    return dp[m][n];
}
```

```cpp
// Optimized 1DP
int uniquePaths(int m, int n) {
    vector<int> dp(n, 1); // base case: first row is all 1s
    for (int row = 1; row < m; ++row) {
        for (int col = 1; col < n; ++col) {
            dp[col] = dp[col] + dp[col - 1];
        }
    }
    return dp[n - 1];
}
```

Solution – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Solution 3 (combinatorics)

- **Grid size:** m x n
- **Start:** top-left cell (0, 0)
- **End:** bottom-right cell (m – 1, n – 1)
- To get from the top-left to the bottom-right:

  You must move exactly m - 1 times down

  And exactly n - 1 times right

  These two types of moves must be made in some order, with a total of:

$$(m - 1) + (n - 1) = m + n - 2 \ moves$$

- **Hence,** from a sequence of m + n - 2 moves, choose m - 1 of them to be down moves (the rest will be right), or vice versa

$$\boldsymbol{Number\ of\ unique\ paths} = \binom{\boldsymbol{m+n-2}}{\boldsymbol{m-1}} = \frac{(\boldsymbol{m+n-2})!}{(\boldsymbol{m-1})!\,(\boldsymbol{n-1})!}$$

Code – 62. Unique Paths

Medium

LeetCode https://leetcode.com/problems/unique-paths

Code

Time: **O(min(m,n))** Space: **O(1)**

```
int uniquePaths(int m, int n) {
    // we will compute the binomial coefficient:
    // (m + n - 2) choose (m - 1) => total moves choose down moves
    // = (m + n - 2)! / ((m - 1)! * (n - 1)!)

    long long res = 1;

    // we compute the result iteratively to avoid large factorials
    // res = (n) * (n+1) * ... * (m+n-2) / (1 * 2 * ... * (m - 1))

    for (int i = 1; i <= m - 1; ++i) {
        // multiply numerator: (n - 1 + i)
        // divide by denominator: i
        res = res * (n - 1 + i) / i;
    }

    return (int)res;
}
```

Problem – 983. Minimum Cost For Tickets

Medium

LeetCode leetcode.com/problems/minimum-cost-for-tickets

Problem

- You are given two arrays of integers, days and costs
- Days represent

Problem – 983. Minimum Cost For Tickets

Medium

LeetCode leetcode.com/problems/minimum-cost-for-tickets

Solution

- …

Problem – 983. Minimum Cost For Tickets

Medium

LeetCode leetcode.com/problems/minimum-cost-for-tickets

Code

Time: **O(-)** Space: **O(-)**

- …

EOF

Tips

Problem Statement / Solution / Code Time: **O(n)** Space: **O(n)**

- …

Problem – number. name

Easy Hard Medium

LeetCode leetcode.com/problems/...

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- 1
- 2