Algorithm and Problem Solving Cheatsheet in C++

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

Rui F. David, 2024

rfdavid.com

MOTIVATION

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 learned 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***

Useful links

https://takeuforward.org/interviews/blind-75-leetcode-problems-detailed-video-solutions

ARRAY

Arrays

Characteristics

- **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.

Arrays – Kadane’s algorithm

Problem - Best Time to Buy and Sell Stock

Easy

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

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

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

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

Example 1

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

Output: 5

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

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

Example 2

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

Output: 0

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

Solution - Best Time to Buy and Sell Stock

Easy

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

```
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;
}
```

BINARY

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

  1 $+ 1 = 0$ (subtract)

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

Negabinary

Example 1

```
  11 1
   1011
 + 1110
------
 = 110001
```

red 1 = negative carry

green 1 = regular carry

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

Example 2

```
  1111
  101010
+ 101100
--------
= 11110110
```

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 (DFS)

Problem - Keys and Rooms

Medium

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;
}
```

GRAPH (BFS)

Problem – Maximum Level Sum of a Binary Tree

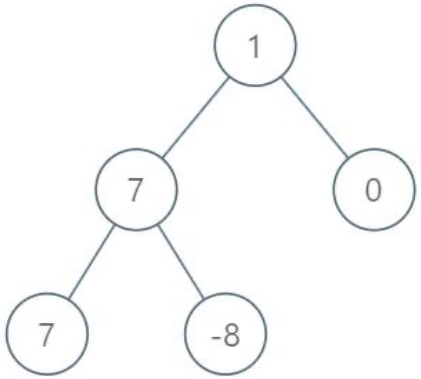
Medium

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

Given the **root** of a binary tree, the level of its root is **1**, the level of its children is **2**, and so on.

Return the **smallest level** x such that the sum of all the values of nodes at level **x** is **maximal.**

Input: root = [1,7,0,7,-8,null,null]
Output: 2
Explanation:
Level 1 sum = 1.
Level 2 sum = 7 + 0 = 7.
Level 3 sum = 7 + -8 = -1.
So we return the level with the maximum sum which is level 2.



Solution – Maximum Level Sum of a Binary Tree

Medium

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

```
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;
}
```

TREE

Node Traversal

In-order etc

Problem – Maximum Depth of Binary Tree

Easy

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

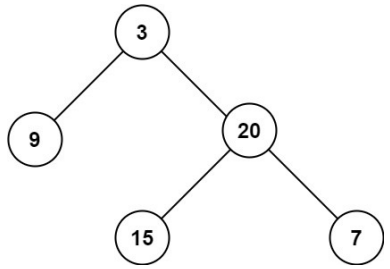
Given the **root** of a binary tree, return its maximum depth.

A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1

Input: root = [3,9,20,null,null,15,7]

Output: 3



Example 2

Input: root = [1,null,2]

Output: 2

Solution – Maximum Depth of Binary Tree

Easy

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

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

Problem – Kth Smallest Element in a BST

Medium

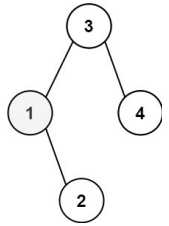
https://leetcode.com/problems/kth-smallest-element-in-a-bst

Given the **root** of a binary search tree, and an integer **k**, return the **k**th smallest value (1-indexed) of all the values of the nodes in the tree.

Example 1

Input: root = [3,1,4,null,2], k = 1

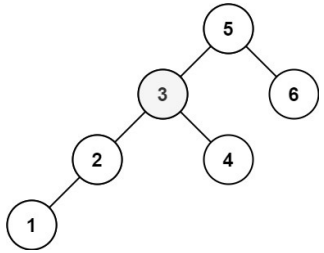
Output: 1



Example 2

Input: root = [5,3,6,2,4,null,null,1], k = 3

Output: 3



Solution – Maximum Depth of Binary Tree

Medium

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

```
int kthSmallest(TreeNode* root, int k) {
    int count = 0;
    int output;
    traverse(root, count, output, k);
    return output;
}

// perform in-order traversal:  left, node, right
void traverse(TreeNode* node, int& count, int &output, int k) {
    if (!node) return;
    traverse(node->left, count, output, k);
    count++;
    if (count == k) {
        output = node->val;
        return;
    }
    traverse(node->right, count, output, k);
}
```

LINKED LIST

Problem – Swap Nodes in Pair

Medium

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

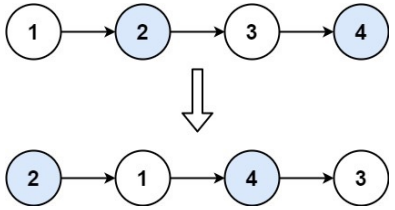
Problem

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

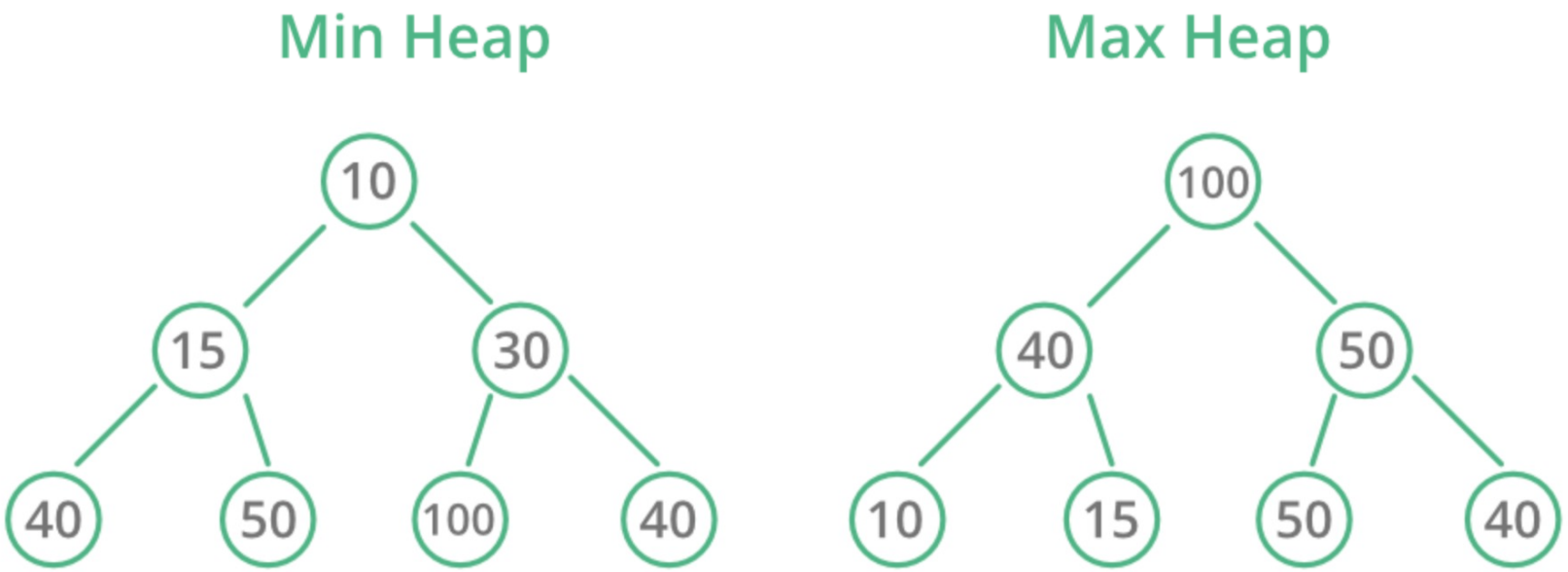
https://leetcode.com/problems/swap-nodes-in-pairs

```
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

```cpp
// 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

```cpp
// 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();
}
```

DYNAMIC PROGRAMMING

Dynamic Programming

Dynamic Programming (DB) 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.

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

Blind 75 | Easy

LeetCode

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

Blind 75

Easy

LeetCode

```cpp
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];
}
```