

# Leetcode Solutions

SVR

August 14, 2025

# Introduction

Following are my solutions for some leetcode problems. The solutions and code are primarily in C++ owing to the fact that I'm already using Python in my research, and C++ for the engineering part. However, C++ is something I'm trying to go deeper owing to the fact that I'm improving my ability to build low latency systems, which primarily use C/C++.

# Template Script

## Description

The following script is forked each time I want to locally work on a leetcode problem. The subsequent solutions in the later sections also have the functions present in this particular script in their scope. So this script also serves to provide an idea as to the functions, and what not, that are available. Note that the standard practice is to have these functions written in another file and have it included in the main script. However, I often tinker with these functions based on the problem at hand. Thus, the not-so-standard approach.

## Template.cpp

---

```
1 using std::map;
2 using std::format;
3 using std::deque;
4 using std::pair;
5
6 // vector printing function
7 template<typename T>
8 void fPrintVector(vector<T> input){
9     for(auto x: input) cout << x << ", ";
10    cout << endl;
11 }
12
13 template<typename T>
14 void fPrintMatrix(vector<T> input){
15     for(auto x: input){
16         for(auto y: x){
17             cout << y << ", ";
18         }
19         cout << endl;
20     }
```

```

21 }
22
23 template<typename T, typename T1>
24 void fPrintHashmap(unordered_map<T, T1> input){
25     for(auto x: input){
26         cout << format("[{},{}] \n", x.first, x.second);
27     }
28     cout <<endl;
29 }
30
31 struct TreeNode {
32     int val;
33     TreeNode *left;
34     TreeNode *right;
35     TreeNode() : val(0), left(nullptr), right(nullptr) {}
36     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
37     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
38 };
39
40
41 struct ListNode {
42     int val;
43     ListNode *next;
44     ListNode() : val(0), next(nullptr) {}
45     ListNode(int x) : val(x), next(nullptr) {}
46     ListNode(int x, ListNode *next) : val(x), next(next) {}
47 };
48
49 void fPrintBinaryTree(TreeNode* root){
50     // sending it back
51     if (root == nullptr) return;
52
53     // printing

```

```

54 PRINTLINE
55 cout << "root->val = " << root->val << endl;
56
57 // calling the children
58 fPrintBinaryTree(root->left);
59 fPrintBinaryTree(root->right);
60
61 // returning
62 return;
63
64 }
65
66 void fPrintLinkedList(ListNode* root){
67     if (root == nullptr) return;
68     cout << root->val << ", ";
69     fPrintLinkedList(root->next);
70     return;
71 }
72
73 template<typename T>
74 void fPrintContainer(T input){
75     for(auto x: input) cout << x << ", ";
76     cout << endl;
77     return;
78 }
79
80 struct Stopwatch
81 {
82     std::chrono::time_point<std::chrono::high_resolution_clock> startpoint;
83     std::chrono::time_point<std::chrono::high_resolution_clock> endpoint;
84     std::chrono::duration<long long, std::nano> duration;
85
86     // constructor

```

```

87  Stopwatch()    {startpoint = std::chrono::high_resolution_clock::now();}
88  void start()   {startpoint = std::chrono::high_resolution_clock::now();}
89  void stop()    {endpoint   = std::chrono::high_resolution_clock::now(); fetchtime();}
90
91  void fetchtime(){
92      duration = std::chrono::duration_cast<std::chrono::nanoseconds>(endpoint - startpoint);
93      cout << format("{} nanoseconds \n", duration.count());
94  }
95  void fetchtime(string stringarg){
96      duration = std::chrono::duration_cast<std::chrono::nanoseconds>(endpoint - startpoint);
97      cout << format("{} took {} nanoseconds \n", stringarg, duration.count());
98  }
99 };
100
101
102 // main-file =====
103 int main(){
104
105     // input- configuration
106
107
108
109
110     // return
111     return(0);
112
113 }

```

---

# 1. Two Sum

## Question

Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.

## Solution

## Code

---

```
1 int main(){
2
3     // input- configuration
4     vector<int> nums {2, 7, 11, 15};
5     int target {9};
6
7     // setup
8     int complement {0};
9     unordered_map<int, int> number_to_index;
10    vector<int> finaloutput;
11
12    // filling the unordered_map
13    for(int i = 0; i < nums.size(); ++i){
14
15        // calculating complement
16        complement = target - nums[i];
17
18        // checking if complement is present in registry
19        if(number_to_index.find(complement) != number_to_index.end()) [[unlikely]]
20        {
```

```

21     finaloutput.push_back(number_to_index[complement]); // adding first index
22     finaloutput.push_back(i);                          // adding second index
23     break;                                              // breaking out
24 }
25 else [[likely]]
26 {
27     // check if current element is present
28     if (number_to_index.find(nums[i]) == number_to_index.end()) [[likely]]
29     {
30         // adding the [number, index] pair to the hashmap
31         number_to_index[nums[i]] = i;
32     }
33     else [[unlikely]]
34     {
35         // we'll do nothing since the number and its index is already present
36         continue;
37     }
38 }
39 }
40
41 // printing the final output
42 for(const auto& x : finaloutput) {cout << x << ", ";} cout << endl;
43
44 // return
45 return(0);
46
47 }

```

---



## 392. Is Subsequence

### Question

Given two strings *s* and *t*, return true if *s* is a subsequence of *t*, or false otherwise.

A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

### Code

---

```
1 int main(){
2
3     // input- configuration
4     string s {"abc"};
5     string t {"ahbgdc"};
6
7     // setup
8     int i = 0;
9
10    // going through the elements
11    for(auto x: t) if (x == s[i]) ++i;
12
13    // returning
14    cout << format("final-output = {}\n", static_cast<bool>(i == s.size())) ;
15
16
17    // return
18    return(0);
19
20 }
```

---