Leetcode Solutions

SVR

August 15, 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

```cpp
using std::map;
using std::format;
using std::deque;
using std::pair;

// vector printing function
template<typename T>
void fPrintVector(vector<T> input){
    for(auto x: input) cout << x << ",";
    cout << endl;
}

template<typename T>
void fPrintMatrix(vector<T> input){
    for(auto x: input){
        for(auto y: x){
            cout << y << ",";
        }
        cout << endl;
    }
```

```
}

template<typename T, typename T1>
void fPrintHashmap(unordered_map<T, T1> input){
    for(auto x: input){
        cout << format("[{},{}] \n", x.first, x.second);
    }
    cout <<endl;
}

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


struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x), next(next) {}
};

void fPrintBinaryTree(TreeNode* root){
    // sending it back
    if (root == nullptr) return;

    // printing
    PRINTLINE
    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);
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
```

```
    void fetchtime(){
        duration = std::chrono::duration_cast<std::chrono::nanoseconds>(endpoint - startpoint);
        cout << format("{} nanoseconds \n", duration.count());
    }
    void fetchtime(string stringarg){
        duration = std::chrono::duration_cast<std::chrono::nanoseconds>(endpoint - startpoint);
        cout << format("{} took {} nanoseconds \n", stringarg, duration.count());
    }
};


// main-file =========================================================================
int main(){

    // input- configuration




    // return
    return(0);

}
```

1. Two Sum

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

Code

```cpp
int main(){

    // input- configuration
    vector<int> nums  {2, 7, 11, 15};
    int        target {9};

    // setup
    int                   complement    {0};
    unordered_map<int, int> number_to_index;
    vector<int>           finaloutput;

    // filling the unordered_map
    for(int i = 0; i < nums.size(); ++i){

        // calculating complement
        complement = target - nums[i];

        // checking if complement is present in registry
        if(number_to_index.find(complement) != number_to_index.end()) [[unlikely]]
        {
            finaloutput.push_back(number_to_index[complement]); // adding first index
            finaloutput.push_back(i);                           // adding second index
            break;                                              // breaking out
        }
```

```
        else [[likely]]
        {
            // check if current element is present
            if (number_to_index.find(nums[i]) == number_to_index.end()) [[likely]]
            {
                // adding the [number, index] pair to the hashmap
                number_to_index[nums[i]] = i;
            }
            else [[unlikely]]
            {
                // we'll do nothing since the number and its index is already present
                continue;
            }
        }
    }

    // printing the final output
    for(const auto& x : finaloutput) {cout << x << ", ";} cout << endl;

    // return
    return(0);

}
```

2. Add Two Numbers

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list. You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Code

```
int main(){

    // input- configuration
    ListNode* l1   = new ListNode(2);
    l1->next       = new ListNode(4);
    l1->next->next = new ListNode(3);

    ListNode* l2   = new ListNode(5);
    l2->next       = new ListNode(6);
    l2->next->next = new ListNode(4);

    // setup
    ListNode* traveller_1 = l1;
    ListNode* traveller_2 = l2;
    ListNode* finalOutput = new ListNode(-1);
    ListNode* traveller_fo = finalOutput;

    int sum               {0};
    int carry             {0};
    int value_1           {0};
    int value_2           {0};

    // moving through the two nodes
    while(traveller_1 != nullptr || traveller_2 != nullptr){
```

```
25
26          // adding the two numbers
27          value_1 = traveller_1 == nullptr ? 0 : traveller_1->val;
28          value_2 = traveller_2 == nullptr ? 0 : traveller_2->val;
29
30          // calculating sum
31          sum     = value_1 + value_2 + carry;
32          if (sum >= 10) [[unlikely]]  {sum -= 10; carry = 1;}
33          else           [[likely]]    {carry = 0;}
34
35          // creating node
36          traveller_fo->next = new ListNode(sum);
37          traveller_fo       = traveller_fo->next;
38
39          // updating the two pointers
40          if(traveller_1 != nullptr) [[likely]] {traveller_1 = traveller_1->next;}
41          if(traveller_2 != nullptr) [[likely]] {traveller_2 = traveller_2->next;}
42      }
43
44      // creating a final node if carry is non-zero
45      if (carry == 1) [[unlikely]] {
46          traveller_fo->next = new ListNode(carry);
47      }
48
49      // printing the final output
50      traveller_fo = finalOutput->next;
51      cout << format("final-output = ");
52      while(traveller_fo != nullptr){
53          cout << traveller_fo->val << ", ";
54          traveller_fo = traveller_fo->next;
55      }
56      cout << "\n";
57
58      // return
59      return(0);
```

```
60
61 }
```

11. Container with most water

- You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).
- Find two lines that together with the x-axis form a container, such that the container contains the most water.
- Return the maximum amount of water a container can store.
- Notice that you may not slant the container.

Code

```
int main(){

    // input- configuration
    vector<int> height {1,8,6,2,5,4,8,3,7};

    // setup
    int left      {0};
    int right     {static_cast<int>(height.size())-1};
    int maxvolume {-1};
    int currvolume {-1};

    // two-pointer approach
    while(left < right){

        // calculating volumes
        currvolume = (right - left) * std::min(height[left], height[right]);
        maxvolume  = maxvolume > currvolume ? maxvolume : currvolume;

        // adjusting left and right based on volume
```

```
        if (height[left] < height[right]) {++left;}
        else                              {--right;}
    }

    // printing
    cout << format("maxvolume = {}\n", maxvolume);

    // return
    return(0);

}
```

88. Merge Sorted Array

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

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

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

Code

```
int main(){

    // input- configuration
    vector<int> nums1 {1, 2, 3, 0, 0, 0};
    vector<int> nums2 {2, 5, 6};
    int m  {3};
    int n  {3};

    // setup
    int p1     {m-1};
    int p2     {n-1};
    int p3     {m+n-1};

    int curr1  {-1};
    int curr2  {-1};

    // going the other way
    while(p1 >= 0 || p2 >= 0)
    {
```

```
        // printing the values
        curr1 = p1 >= 0 ? nums1[p1] : std::numeric_limits<int>::min();
        curr2 = p2 >= 0 ? nums2[p2] : std::numeric_limits<int>::min();

        // assigning value
        if (curr1 > curr2) {nums1[p3] = curr1; --p3; --p1;}
        else               {nums1[p3] = curr2; --p3; --p2;}

    }

    // printing the final output
    cout << format("finaloutput = "); fPrintVector(nums1);

    // return
    return(0);
}
```

392. Is Subsequence

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

```
int main(){

    // input- configuration
    string s   {"abc"};
    string t   {"ahbgdc"};

    // setup
    int i = 0;

    // going through the elements
    for(auto x: t) if (x == s[i]) ++i;

    // returning
    cout << format("final-output = {}\n", static_cast<bool>(i == s.size())) ;


    // return
    return(0);

}
```