Leetcode Solutions

SVR

July 27, 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;

    // calling the children
    fPrintBinaryTree(root->left);
    fPrintBinaryTree(root->right);

    // returning
    return;

}

void fPrintLinkedList(ListNode* root){
    if (root == nullptr) return;
    cout << root->val << ", ";
    fPrintLinkedList(root);
    return;
}

template<typename T>
void fPrintContainer(T input){
    for(auto x: input) cout << x << ", ";
    cout << endl;
    return;
}

struct StopWatch
{
    std::chrono::time_point<std::chrono::high_resolution_clock> startpoint;
    std::chrono::time_point<std::chrono::high_resolution_clock> endpoint;
    std::chrono::duration<long long, std::nano>           duration;

    // constructor
    StopWatch()   {startpoint = std::chrono::high_resolution_clock::now();}
```

```cpp
    void start()  {startpoint = std::chrono::high_resolution_clock::now();}
    void stop()   {endpoint  = std::chrono::high_resolution_clock::now(); fetchtime();}

    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);

}
```

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).

Solution

This is a fairly easy coding question. The main idea is to iterate through string t and compare each character to the current character in string s. If a match is found, you increment the `index_s` pointer to check the next character in s. Regardless of whether a match occurs, the `index_t` pointer is always incremented, ensuring that the loop progresses through the entire string t.

The loop continues until either all characters in s have been matched in order (meaning s is a subsequence of t), or string t has been fully traversed without completing the match. After the loop, you simply check if `index_s` has reached the end of string s. If it has, you return true to indicate a successful subsequence match; otherwise, you return false.

Code

```cpp
int main(){

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

    // setting up pointers
    int index_s       {0};
    int index_t       {0};

    // going throuh
    while(index_t < t.size()){

        // checking if pointer-s has already finished
        if (index_s >= s.size()) {break;}

        // comparing the two values
        if (s[index_s] == t[index_t++]) {index_s++;}
    }

    // checking whether it is a subsequence or not
    if (index_s == s.size()) {return true;}
    else                     {return false;}
}
```