Leetcode Solutions

SVR

August 29, 2025

Contents

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
// including header-files
#include <algorithm>
#include <unordered_set>
#include <bitset>
#include <climits>
#include <cstddef>
#include <iostream>
#include <limits>
#include <map>
#include <new>
#include <stdlib.h>
#include <unordered_map>
#include <vector>
#include <set>
#include <numeric>
#include <functional>


// hash-deinfes
#define PRINTSPACE std::cout << "\n\n\n\n" << std::endl;
```

```cpp
#define PRINTLINE std::cout << "==============================================" << std::endl;

// borrowing from namespace std
using std::cout;
using std::endl;
using std::vector;
using std::string;
using std::unordered_map;
using std::map;
using std::format;
using std::deque;
using std::pair;
using std::min;
using std::max;

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

template<typename T>
void fpv(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("[{},{}] | ", 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(string prefix,
                      ListNode* root){
    if (root == nullptr) return;
    cout << prefix;
    std::function<void(ListNode*)> runlinkedlist = [&runlinkedlist](ListNode* root){
        if (root == nullptr) return;
        cout << root->val << " -> ";
        runlinkedlist(root->next);
    };
    runlinkedlist(root);
    cout << "|" << endl;
    return;
}

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

struct Timer
{
    std::chrono::time_point<std::chrono::high_resolution_clock> startpoint;
```

```cpp
    std::chrono::time_point<std::chrono::high_resolution_clock> endpoint;
    std::chrono::duration<long long, std::nano>           duration;

    // constructor
    Timer()       {startpoint = std::chrono::high_resolution_clock::now();}
    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());
    }
    void measure(){
        auto temp = std::chrono::high_resolution_clock::now();
        auto nsduration = std::chrono::duration_cast<std::chrono::nanoseconds>(temp - startpoint);
        auto msduration = std::chrono::duration_cast<std::chrono::microseconds>(temp - startpoint);
        auto sduration  = std::chrono::duration_cast<std::chrono::seconds>(temp - startpoint);
        cout << format("{} nanoseconds | {} microseconds | {} seconds \n",
            nsduration.count(), msduration.count(), sduration.count());
    }
    ~Timer(){
        measure();
    }
};

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

    // starting timer
    Timer timer;

```

```
    // input- configuration


    // setup






    // 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.

Examples

1. **Example 1:**
    - Input: nums = [2,7,11,15], target = 9
    - Output: [0,1]
    - Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
2. **Example 2:**
    - Input: nums = [3,2,4], target = 6
    - Output: [1,2]
3. **Example 3:**
    - Input: nums = [3,3], target = 6
    - Output: [0,1]

Constraints:

- $2 \leq \text{nums.length} \leq 10^4$

- $-10^9 \leq \text{nums[i]} \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$
- Only one valid answer exists.

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.

Examples

1. **Example 1:**
    - Input: l1 = [2,4,3], l2 = [5,6,4]
    - Output: [7,0,8]
    - Explanation: 342 + 465 = 807.
2. **Example 2:**
    - Input: l1 = [0], l2 = [0]
    - Output: [0]
3. **Example 3:**
    - Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
    - Output: [8,9,9,9,0,0,0,1]

Constraints:

- The number of nodes in each linked list is in the range [1, 100].

- $0 \leq$ Node.val ≤ 9
- It is guaranteed that the list represents a number that does not have leading zeros.

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){

        // adding the two numbers
        value_1 = traveller_1 == nullptr ? 0 : traveller_1->val;
```

```
        value_2 = traveller_2 == nullptr ? 0 : traveller_2->val;

        // calculating sum
        sum    = value_1 + value_2 + carry;
        if (sum >= 10) [[unlikely]]  {sum -= 10; carry = 1;}
        else          [[likely]]     {carry = 0;}

        // creating node
        traveller_fo->next = new ListNode(sum);
        traveller_fo      = traveller_fo->next;

        // updating the two pointers
        if(traveller_1 != nullptr) [[likely]] {traveller_1 = traveller_1->next;}
        if(traveller_2 != nullptr) [[likely]] {traveller_2 = traveller_2->next;}
    }

    // creating a final node if carry is non-zero
    if (carry == 1) [[unlikely]] {
        traveller_fo->next = new ListNode(carry);
    }

    // printing the final output
    traveller_fo = finalOutput->next;
    cout << format("final-output = ");
    while(traveller_fo != nullptr){
        cout << traveller_fo->val << ", ";
        traveller_fo = traveller_fo->next;
    }
    cout << "\n";

    // return
    return(0);

}
```

3. Longest Substring Without Repeating Characters

Given a string s, find the length of the longest substring without duplicate characters.

1. **Example 1:**
   - Input: s = ”abcabcbb”
   - Output: 3
   - Explanation: The answer is ”abc”, with the length of 3.
2. **Example 2:**
   - Input: s = ”bbbbb”
   - Output: 1
   - Explanation: The answer is ”b”, with the length of 1.
3. **Example 3:**
   - Input: s = ”pwwkew”
   - Output: 3
   - Explanation: The answer is ”wke”, with the length of 3. Notice that the answer must be a substring, ”pwke” is a subsequence and not a substring.

Code

```cpp
int main(){

    // input- configuration
    string s   {"tmmzuxt"};

    // setup
    unordered_map<char, int> histogram;
    int p1 {0};
    char curr;
    int finaloutput {-1};
    int temp_length {-1};

    // going through the thing
    for(int p2 = 0; p2<s.size(); ++p2){

        // moving to another variable
        curr = s[p2];

        // checking if current character is in histogram
        if (histogram.find(curr) == histogram.end()) [[unlikely]]
        {
            histogram[curr] = 1;
        }
        else [[likely]]
        {
            // checking if count is zero
            if (histogram[curr] == 0)
            {
                histogram[curr] = 1;
            }
            else
            {
                // moving p1 until it arrives at first instance of curr
                while(s[p1] != curr)
```

```
                {
                    --histogram[s[p1]];
                    ++p1;
                }
                ++p1;
                histogram[curr] = 1;
            }
        }

        // calculating longest length
        finaloutput = finaloutput > (p2-p1+1) ? finaloutput : (p2-p1+1);
    }

    // printing
    cout << format("longest length = {} \n", finaloutput);

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

4. Median Of Two Sorted Array

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

Examples

1. **Example 1:**
    - Input: nums1 = [1,3], nums2 = [2]
    - Output: 2.00000
    - Explanation: merged array = [1,2,3] and median is 2.
2. **Example 2:**
    - Input: nums1 = [1,2], nums2 = [3,4]
    - Output: 2.50000
    - Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

Constraints:

1. nums1.length == m
2. nums2.length == n
3. $0 \leq m \leq 1000$

4. $0 \leq n \leq 1000$
5. $1 \leq m + n \leq 2000$
6. $-10^6 \leq nums1[i], nums2[i] \leq 10^6$

Code

```
int main(){

    // input- configuration
    vector<int> nums1 {1, 2};
    vector<int> nums2 {3, 4};


    // setup
    vector<int>& first = nums1[0] <= nums2[0] ? nums1 : nums2;
    vector<int>& second = nums1[0] > nums2[0] ? nums1 : nums2;
    int left_first {0};
    int right_first {static_cast<int>(first.size())-1};
    int left_second {0};
    int right_second  {static_cast<int>(second.size())-1};
    int left_value = first[left_first] < second[left_second] ? first[left_first] : second[left_second];
    int right_value = first[right_first] > second[right_second] ? first[right_first] : second[right_second];
    int numiterations {static_cast<int>((nums1.size() + nums2.size())/2)};


    // running for a certain number of iterations
    for(int i = 0; i<numiterations+1; ++i){

        // updating left
        if (first[left_first] < second[left_second]) {left_value = first[left_first]; ++left_first;}
        else                                        {left_value = second[left_second]; ++left_second;}
```

```
        if (first[right_first] > second[right_second]) {right_value = first[right_first]; --right_first;}
        else                                          {right_value = second[right_second]; --right_second;}

        // printing
        cout << format("left-value = {}, right-value = {}\n", left_value, right_value);
    }

    cout << format("median = {}\n", static_cast<double>(left_value + right_value)/2.0);



    // return
    return(0);

}
```

6. Zigzag Conversion

The string ”PAYPALISHIRING” is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

P	-	A	-	H	-	N
A	P	L	S	I	I	G
Y	-	I	-	R	-	-

And then read line by line: “PAHNAPLSIIGYIR”

Examples

1. **Example 1:**
   - Input: s = “PAYPALISHIRING”, numRows = 3
   - Output: “PAHNAPLSIIGYIR”
2. **Example 2:**
   - Input: s = “PAYPALISHIRING” , numRows = 4
   - Output: “PINALSIGYAHRPI”
3. **Example 3:**
   - Input: s = “A”, numRows = 1
   - Output: “A”

Constraints:

1. $1 \leq \text{s.length} \leq 1000$
2. s consists of English letters (lower-case and upper-case), ',' and '.'.
3. $1 \leq \text{numRows} \leq 1000$

Code

```cpp
int main(){

    // input- configuration
    string s   {"PAYPALISHIRING"};
    int numRows {4};

    // trivial case
    if (numRows == 1) {cout << format("finaloutput = {}\n", s); return 0;}

    // setup
    int modlength {2*numRows-2};
    int numblocks {(static_cast<int>(s.size())+ modlength-1)/modlength};
    int sourceindex {-1};
    string finaloutput;

    // going through the thing
    for(int row = 0; row < numRows; ++row){
        for(int i = 0; i<numblocks; ++i){

            // first column of each block
            sourceindex = row + modlength * i;
            if (sourceindex<s.size())       {finaloutput += s[sourceindex];}

```

```
            // continuing in case of boundary rows
            if (row == 0 || row == numRows-1) {continue;}

            // taking care of the case where non-boundary rows
            sourceindex = modlength - row + modlength*i;
            if (sourceindex < s.size())      {finaloutput += s[sourceindex];}
        }
    }

    // printing the final output
    cout << format("final-output = {}\n", finaloutput);


    // return
    return(0);

}
```

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.

Examples

1. **Example 1:**
    - Input: height = [1,8,6,2,5,4,8,3,7]
    - Output: 49
    - Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.
2. **Example 2:**
    - Input: height = [1,1]
    - Output: 1

Constraints

- n == height.length
- $2 \leq n \leq 10^5$
- $0 \leq height[i] \leq 10^4$

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);

}
```

12. Integer to Roman

Roman numerals are formed by appending the conversions of decimal place values from highest to lowest. Converting a decimal place value into a Roman numeral has the following rules:

- If the value does not start with 4 or 9, select the symbol of the maximal value that can be subtracted from the input, append that symbol to the result, subtract its value, and convert the remainder to a Roman numeral.
- If the value starts with 4 or 9 use the subtractive form representing one symbol subtracted from the following symbol, for example, 4 is 1 (I) less than 5 (V): IV and 9 is 1 (I) less than 10 (X): IX. Only the following subtractive forms are used: 4 (IV), 9 (IX), 40 (XL), 90 (XC), 400 (CD) and 900 (CM).
- Only powers of 10 (I, X, C, M) can be appended consecutively at most 3 times to represent multiples of 10. You cannot append 5 (V), 50 (L), or 500 (D) multiple times. If you need to append a symbol 4 times use the subtractive form.

Given an integer, convert it to a Roman numeral.

Examples

1. **Example 1**
    - Input: num = 3749
    - Output: "MMMDCCXLIX"
    - Explanation:
        - 3000 = MMM as 1000 (M) + 1000 (M) + 1000 (M)
        - 700 = DCC as 500 (D) + 100 (C) + 100 (C)
        - 40 = XL as 10 (X) less of 50 (L)
        - 9 = IX as 1 (I) less of 10 (X)

- Note: 49 is not 1 (I) less of 50 (L) because the conversion is based on decimal places

2. **Example 2:**
    - Input: num = 58
    - Output: ”LVIII”
    - Explanation:
        - 50 = L
        - 8 = VIII
3. **Example 3:**
    - Input: num = 1994
    - Output: ”MCMXCIV”
    - Explanation:
        - 1000 = M
        - 900 = CM
        - 90 = XC
        - 4 = IV

Constraints

- $1 \leq \text{num} \leq 3999$

Code

```
int main(){

    // input- configuration
    int num    {1994};

    // setup
    vector<pair<int, string>> numToString {
        {1, "I"},
        {4, "IV"},
        {5, "V"},
        {9, "IX"},
        {10, "X"},
        {40, "XL"},
        {50, "L"},
        {90, "XC"},
        {100, "C"},
        {400, "CD"},
        {500, "D"},
        {900, "CM"},
        {1000, "M"}
    };                                                                          // number-string pairs
    string finaloutput;                                                         // variable to hold the final output
    int    count;                                                               // variable that will hold the counts
    auto mulstring = [](const int& count,
                        const string& inputstring,
                        string& finaloutput){
        if (count == 0) {return;}
        for(int i = 0; i<count; ++i){finaloutput += inputstring;}
    };                                                                          // lambda-function for int * string
         multiplications (python style)

    // going through the hashmap from the end
    for(int i = numToString.size()-1; i>=0; --i){

```

```
        // calculating count
        count   = num / numToString[i].first;
        num     = num - numToString[i].first*count;

        // adding to final output
        mulstring(count, numToString[i].second, finaloutput);
    }

    // printing the final-output
    cout << format("finaloutput = {}\n", finaloutput);

    // return
    return(0);

}
```

13. Roman To Integer

Roman numerals are represented by seven different symbols: I(1), V(5), X(10), L(50), C(100), D(500) and M(1000). For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II. Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

1. I can be placed before V (5) and X (10) to make 4 and 9.
2. X can be placed before L (50) and C (100) to make 40 and 90.
3. C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Examples

1. **Example 1**
   - Input: s = ”III”
   - Output: 3
   - Explanation: III = 3.
2. **Example 2**
   - Input: s = ”LVIII”
   - Output: 58

- Explanation: L = 50, V= 5, III = 3.

3. **Example 3**

    - Input: s = ”MCMXCIV”
    - Output: 1994
    - Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

Constraints

1. $1 \leq$ s.length ≤ 15
2. s contains only the characters (’I’, ’V’, ’X’, ’L’, ’C’, ’D’, ’M’).
3. It is guaranteed that s is a valid roman numeral in the range [1, 3999].

Code

```
int main(){

    // input- configuration
    string s   {"MCMXCIV"};

    // setup
    int finaloutput {0};
    unordered_map<char, int> charToInt {{'I', 1},
                                        {'V', 5},
                                        {'X', 10},
                                        {'L', 50},
                                        {'C', 100},
```

```
                                          {'D', 500},
                                          {'M', 1000}};

    // going through the string
    for(int i = 0; i<s.size(); ++i){
        if ((i+1)<s.size() && charToInt[s[i]] < charToInt[s[i+1]]) {finaloutput -= charToInt[s[i]];}
        else                                                     {finaloutput += charToInt[s[i]];}
    }

    // printing the final output
    cout << format("finaloutput = {}\n", finaloutput);

    // return
    return(0);

}
```

14. Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string ””.

Examples

1. **Example 1:**
    - Input: strs = [”flower”,”flow”,”flight”]
    - Output: ”fl”
2. **Example 2:**
    - Input: strs = [”dog”,”racecar”,”car”]
    - Output: ””
    - Explanation: There is no common prefix among the input strings.

Constraints:

- $1 \leq$ strs.length ≤ 200
- $0 \leq$ strs[i].length ≤ 200
- strs[i] consists of only lowercase English letters if it is non-empty.

Code

```cpp
int main(){

    // input- configuration
    vector<string> strs {
        "flower",
        "flow",
        "flight"
    };

    // setup
    int p             {0};                                          // index-pointer for boundary
    int runcondition  {true};                                       // breaking condition
    string prefix;

    // going through the vector
    while(runcondition){

        // breaking if it doesn't meet first words length
        if (p >= strs[0].size())         {++p; runcondition = false; break;}

        // checking if this candidate
        for(int i = 1; i<strs.size(); ++i){

            // checking if valid
            if (p >= strs[i].size())     {runcondition = false; break;}

            // checking if same
            if (strs[i][p] != strs[0][p]) {runcondition = false; break;}
        }

        // updating p
        ++p;
    }
```

```
    // subsetting and printing the prefix
    prefix = string(strs[0].begin(), strs[0].begin()+p-1);
    cout << format("finaloutput = {}\n", prefix);

    // return
    return(0);

}
```

19. Remove Nth Node From End of List

Given the head of a linked list, remove the nth node from the end of the list and return its head.

Examples

1. **Example 1:**
    - Input: head = [1,2,3,4,5], n = 2
    - Output: [1,2,3,5]
2. **Example 2:**
    - Input: head = [1], n = 1
    - Output: []
3. **Example 3:**
    - Input: head = [1,2], n = 1
    - Output: [1]

Constraints

- The number of nodes in the list is sz.
- $1 \leq \text{sz} \leq 30$
- $0 \leq \text{Node.val} \leq 100$
- $1 \leq \text{n} \leq \text{sz}$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    ListNode* head                   = new ListNode(1);
    head->next                       = new ListNode(2);
    head->next->next                 = new ListNode(3);
    head->next->next->next           = new ListNode(4);
    head->next->next->next->next = new ListNode(5);
    auto n {2};

    // trivial case
    if (!head) {cout << format("final-output = "); fPrintLinkedList(head); cout << endl;}

    // setup
    auto nodecounter  {0};
    auto prehead      {new ListNode};

    prehead->next              = head;
    ListNode* traveller        = head;
    ListNode* delayedTraveller = prehead;

    // going through the list
    while(traveller){
        if (++nodecounter > n) {delayedTraveller = delayedTraveller->next;}
        traveller = traveller->next;
    }

    // reconnecting
    delayedTraveller->next = delayedTraveller->next->next;

```

```
    // sending back
    cout << format("final-output = ");
    fPrintLinkedList(prehead->next); cout << endl;





    // return
    return(0);

}
```

20. Valid Parentheses

Given a string s containing just the characters '(', ')', ", ", '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Examples

1. **Example 1:**
    - Input: s = "()"
    - Output: true
2. **Example 2:**
    - Input: s = "()[]"
    - Output: true
3. **Example 3:**
    - Input: s = "(]"
    - Output: false

Constraints

- $1 \leq$ s.length $\leq 10^4$
- s consists of parentheses only '()[]'.

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    string s   {"()[]{}"};

    // seutp
    deque<char> var00;

    // going through the string
    for(auto x: s){
        // pushing or poppping
        if (x == '(' || x == '[' || x == '{') {var00.push_back(x);}
        else{
            if (var00.size()==0)              {cout << format("final-output = false\n"); return 0;}
            if (x == ')' && var00.back() == '(') {var00.pop_back(); continue;}
            if (x == ']' && var00.back() == '[') {var00.pop_back(); continue;}
            if (x == '}' && var00.back() == '{') {var00.pop_back(); continue;}
            cout << format("final-output = false\n"); return 0;
        }
    }

    // checking if anything is left
```

```
    if (var00.size()!=0) {cout << format("final-output = false\n"); return 0;}


    // true-case
    cout << format("final-output = true\n"); return 0;

    // return
    return(0);

}
```

21. Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

Examples

1. **Example 1:**
   - Input: list1 = [1,2,4], list2 = [1,3,4]
   - Output: [1,1,2,3,4,4]
2. **Example 2:**
   - Input: list1 = [], list2 = []
   - Output: []
3. **Example 3:**
   - Input: list1 = [], list2 = [0]
   - Output: [0]

Constraints

- The number of nodes in both lists is in the range [0, 50].

- -100 $\leq$ Node.val $\leq$ 100
- Both list1 and list2 are sorted in non-decreasing order.

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    ListNode* list1   = new ListNode(1);
    list1->next       = new ListNode(2);
    list1->next->next = new ListNode(4);

    ListNode* list2   = new ListNode(1);
    list2->next       = new ListNode(3);
    list2->next->next = new ListNode(4);

    // setup
    ListNode* finalOutput = new ListNode;
    ListNode* traveller_final = finalOutput;
    ListNode* traveller_1 = list1;
    ListNode* traveller_2 = list2;

    // going through the two lists
    while (traveller_1 != nullptr && traveller_2 != nullptr) {

        // comparing values
        if (traveller_1->val < traveller_2->val) {
            traveller_final->next = traveller_1;           // linking node to final
            traveller_final       = traveller_final->next; // moving node to new link
```

```
            if (traveller_1->next != nullptr)
                traveller_1           = traveller_1->next;     // updating traveller 1
            else{
                traveller_1 = nullptr;
                break;
            }

        }
        else {
            traveller_final->next = traveller_2;           // linking node to final
            traveller_final       = traveller_final->next; // moving node to new link
            if (traveller_2->next != nullptr)
                traveller_2           = traveller_2->next;     // updating travel 2
            else{
                traveller_2 = nullptr;
                break;
            }
        }
    }

    // checking if anything is left
    while (traveller_1 != nullptr){
        traveller_final->next = traveller_1;
        traveller_final       = traveller_final->next;
        traveller_1           = traveller_1->next;
    }

    while (traveller_2 != nullptr){
        traveller_final->next = traveller_2;
        traveller_final       = traveller_final->next;
        traveller_2           = traveller_2->next;
    }

    // printing the final-output
```

```
    cout << format("finaloutput = ");
    fPrintLinkedList(finalOutput); cout << endl;

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

22. Generate Parentheses

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

Examples

1. **Example 1:**
   - Input: n = 3
   - Output: `["((()))","(()())","(())()","()(())","()()()"]`
2. **Example 2:**
   - Input: n = 1
   - Output: `["()"]`

Constraints

- $1 \leq n \leq 8$

Code

```
void foo(int                numopens,
         int                numcloses,
         const int&         n,
         string             runningstring,
         vector<string>&  finalOutput){
```

```cpp

    // sending it back
    if (numopens > n || numcloses > n) return;
    else if (numopens == n && numcloses == n) {finalOutput.push_back(runningstring); return;}

    // opening route
    foo(numopens+1, numcloses, n, runningstring + "(", finalOutput);

    // closing route
    if (numopens>numcloses)
        foo(numopens, numcloses+1, n, runningstring+")", finalOutput);

    // returning
    return;

}

int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto n {3};

    // trivial case
    if (n == 0) {cout << format("finalOutput = []\n"); return 0;}

    // calling the function
    vector<string> finalOutput;
    foo(0, 0, n, "", finalOutput);

    // returning
    cout << format("finalOutput = {}\n", finalOutput);

```

```
    // return
    return(0);

}
```

25. Reverse Nodes in k-Group

Given the head of a linked list, reverse the nodes of the list k at a time, and return the modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is.

You may not alter the values in the list's nodes, only nodes themselves may be changed.

Examples

1. **Example 1:**
    - Input: head = [1,2,3,4,5], k = 2
    - Output: [2,1,4,3,5]
2. **Example 2:**
    - Input: head = [1,2,3,4,5], k = 3
    - Output: [3,2,1,4,5]

Constraints

- The number of nodes in the list is n.
- $1 \leq k \leq n \leq 5000$
- $0 \leq \text{Node.val} \leq 1000$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    ListNode* head                    = new ListNode(1);
    head->next                        = new ListNode(2);
    head->next->next                  = new ListNode(3);
    head->next->next->next            = new ListNode(4);
    head->next->next->next->next = new ListNode(5);
    auto k     {2};

    // setup
    ListNode* prehead = new ListNode(); prehead->next = head;
    ListNode* traveller = prehead;
    vector<ListNode*> memberList(k, nullptr); memberList.reserve(k);

    // runnings
    while(traveller){
        // filling up the memberList
        int counter {0};
        ListNode* tempotraveller = traveller->next;
        while(counter < k && tempotraveller != nullptr){
            memberList[counter]   = tempotraveller;
            tempotraveller        = tempotraveller->next;
            ++counter;
        }

        // checking breaking condition
        if (counter!=k) break;

        // reconnecting
```

```
        ListNode* beginningOfNextSegment = memberList[memberList.size()-1]->next;

        // reconnecting
        traveller->next = memberList[memberList.size()-1];
        for(int i = memberList.size()-1; i>=1; --i) {memberList[i]->next = memberList[i-1];}
        memberList[0]->next = beginningOfNextSegment;

        // updating traveller
        traveller = memberList[0];
    }

    // returning
    cout << format("final-output = ");
    fPrintLinkedList(prehead->next); cout << endl;

    // return
    return(0);

}
```

26. Remove Duplicates From Sorted Array

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in nums. Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

- Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.
- Return k.

Examples

1. **Example 1:**
   - Input: nums = [1,1,2]
   - Output: 2, nums = `[1,2,_]`
   - Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively. It does not matter what you leave beyond the returned k (hence they are underscores).
2. **Example 2:**
   - Input: nums = [0,0,1,1,1,2,2,3,3,4]
   - Output: 5, nums = `[0,1,2,3,4,_,_,_,_,_]`
   - Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively. It does not matter what you leave beyond the returned k (hence they are underscores).

Constraints:

- $1 \leq \text{nums.length} \leq 3 * 10^4$
- $-100 \leq \text{nums[i]} \leq 100$
- nums is sorted in non-decreasing order.

Code

```cpp
int main(){

    // input- configuration
    vector<int> nums     {1,1};

    // setup
    int p      {0};
    int counter {0};

    // going through the values
    for(int i = 1; i<nums.size(); ++i){

        // check values
        if (nums[i] == nums[p]) {continue;}

        // writing values
        ++p;
        nums[p] = nums[i];
        ++counter;
    }

    // printing the final output
    cout << format("final-output = {}\n", counter+1);
```

```
    cout << format("nums = "); fpv(nums);

    // return
    return(0);

}
```

27. Remove Element

Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The order of the elements may be changed. Then return the number of elements in nums which are not equal to val.

Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:

Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums. Return k.

Examples

1. **Example 1:**

    - Input: nums = [3,2,2,3], val = 3
    - Output: 2, nums = `[2,2,_,_]`
    - Explanation: Your function should return k = 2, with the first two elements of nums being 2. It does not matter what you leave beyond the returned k (hence they are underscores).

2. **Example 2:**

    - Input: nums = [0,1,2,2,3,0,4,2], val = 2
    - Output: 5, nums = `[0,1,4,0,3,_,_,_]`
    - Explanation: Your function should return k = 5, with the first five elements of nums containing 0, 0, 1, 3, and 4. Note that the five elements can be returned in any order. It does not matter what you leave beyond the returned k (hence they are underscores).

Constraints

- $0 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums[i]} \leq 50$
- $0 \leq \text{val} \leq 100$

Code

```
int main(){

    // input- configuration
    vector<int> nums  {0,1,2,2,3,0,4,2};
    int val           {2};

    // setup
    int src       {0};
    int dest      {0};
    int numwrites {0};

    // going through the indices
    while(src < nums.size()){

        // moving the dest until we find a val-position
        while(nums[dest] != val) {++dest;}

        // moving source until we find a non-val position after dest
        src = std::max(src, dest+1);
        while(nums[src] == val)  {++src;};

        // writing
        if (dest < nums.size() && src < nums.size()){
```

```
            nums[dest] = nums[src];
            ++dest;
            ++src;
            ++numwrites;
        }

    }

    // printing the length
    cout << format("updated nums = "); fPrintVector(nums);
    cout << format("finaloutput = {} \n", nums.size()-numwrites-1);

    // return
    return(0);

}
```

28. Find the Index of the First Occurrence in a String

Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Examples

1. **Example 1:**
    - Input: haystack = ”sadbutsad”, needle = ”sad”
    - Output: 0
    - Explanation: ”sad” occurs at index 0 and 6. The first occurrence is at index 0, so we return 0.
2. **Example 2:**
    - Input: haystack = ”leetcode”, needle = ”leeto”
    - Output: -1
    - Explanation: ”leeto” did not occur in ”leetcode”, so we return -1.

Constraints

- $1 \leq$ haystack.length, needle.length $\leq 10^4$
- haystack and needle consist of only lowercase English characters.

Code

```cpp
int main(){

    // input- configuration
    string haystack {"leetcode"};
    string needle  {"leeto"};


    // setup
    int finaloutput {-1};
    auto beginsearch = [haystack, needle](int currindex){
        // starting search
        if(currindex + needle.size() > haystack.size()) {return false;}

        // checking if they're a subset
        for(int i = 0; i<needle.size(); ++i){
            if (haystack[currindex + i] != needle[i]) {return false;}
        }

        return true;
    };

    // going through
    for(int i = 0; i < haystack.size(); ++i){

        // begin search at each index
        auto curroutput = beginsearch(i);

        // writing final output, if a mach
        if (curroutput) {finaloutput = i; break;}

    }

    // printing final output
```

```
    cout << format("final-output = {}\n", finaloutput);

    // return
    return(0);

}
```

36. Valid Sudoku

Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

1. Each row must contain the digits 1-9 without repetition.
2. Each column must contain the digits 1-9 without repetition.
3. Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.

Examples

1. **Example 1**
    - Input: board =
        - `[["5","3",".",".","7",".",".",".","."]`
        - `,["6",".",".","1","9","5",".",".","."]`
        - `,[".","9","8",".",".",".",".","6","."]`
        - `,["8",".",".",".","6",".",".",".","3"]`
        - `,["4",".",".","8",".","3",".",".","1"]`
        - `,["7",".",".",".","2",".",".",".","6"]`
        - `,[".","6",".",".",".",".","2","8","."]`
        - `,[".",".",".","4","1","9",".",".","5"]`
        - `,[".",".",".",".","8",".",".","7","9"]]`
    - Output: true
2. **Example 2:**

- Input: board =
  - `[["8","3",".",".","7",".",".",".","."]`
  - `,["6",".",".","1","9","5",".",".","."]`
  - `,[".","9","8",".",".",".",".","6","."]`
  - `,["8",".",".",".","6",".",".",".","3"]`
  - `,["4",".",".","8",".","3",".",".","1"]`
  - `,["7",".",".",".","2",".",".",".","6"]`
  - `,[".","6",".",".",".",".","2","8","."]`
  - `,[".",".",".","4","1","9",".",".","5"]`
  - `,[".",".",".",".","8",".",".","7","9"]]`
- Output: false
- Explanation: Same as Example 1, except with the 5 in the top left corner being modified to 8. Since there are two 8's in the top left 3x3 sub-box, it is invalid.

Constraints

- board.length == 9
- board[i].length == 9
- board[i][j] is a digit 1-9 or '.'.

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

Code

```cpp
// main-file
int main(){

    // input-configuration
    vector< vector<char> > board;
    board.push_back({'5','3','.','.','7','.','.','.','.'});
    board.push_back({'6','.','.','1','9','5','.','.','.'});
    board.push_back({'.','9','8','.','.','.','.','6','.'});
    board.push_back({'8','.','.','.','6','.','.','.','3'});
    board.push_back({'4','.','.','8','.','3','.','.','1'});
    board.push_back({'7','.','.','.','2','.','.','.','6'});
    board.push_back({'.','6','.','.','.','.','2','8','.'});
    board.push_back({'.','.','.','4','1','9','.','.','5'});
    board.push_back({'.','.','.','.','8','.','.','7','9'});

    // basic method
    int xoffset,  yoffset;
    int row_local, col_local;

    // lambda for converting char to inger
    auto fConvert = [](char x) -> int {
        if (x == '.') return -1;
        else { return static_cast<int>(x - '0');}
    };

    // checking row and column entries
    for(int i = 0; i < 9; ++i){

        // register for each jumn
        vector<int> rowRegister(9, 0);
        vector<int> colRegister(9, 0);
        vector<int> blockRegister(9,0);

```

```
        // going through each jumn
        for(int j = 0; j<9; ++j){

            // along the row
            int var00  = fConvert(board[i][j]);
            if (var00  != -1) {if (++rowRegister[var00-1] > 1) return false;}

            // down the column
            var00      = fConvert(board[j][i]);
            if (var00  !=-1)  {if (++colRegister[var00-1] > 1) return false;}

            // checking block
            row_local  = j / 3;
            col_local  = j % 3;
            xoffset    = i / 3;
            yoffset    = i % 3;

            // calculating registers
            var00 = fConvert(board[3*xoffset + row_local][col_local+3*yoffset]);
            if (var00!=-1)    {if (++blockRegister[var00-1]>1) return false;}
        }
    }

    // returning true
    return true;
}
```

42. Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Examples

1. **Example 1**
   - Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
   - Output: 6
   - Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.
2. **Example 2**
   - Input: height = [4,2,0,3,2,5]
   - Output: 9

Constraints

1. n == height.length
2. $1 \leq n \leq 2 * 10^4$
3. $0 \leq \text{height}[i] \leq 10^5$

Code

```cpp
int main(){

    // input- configuration
    vector<int> height {0,1,0,2,1,0,1,3,2,1,2,1};

    // setup
    vector<int> leftmaxes(height.size(), 0);                                // vector holding biggest-height to left
    vector<int> rightmaxes(height.size(), 0);                               // vector holding biggest-height to the right
    int forwardindex  {0};                                                  // for maintaining forward-index
    int backwardindex {0};                                                  // for maintaining backward-index
    int maxleft       {-1};                                                 // keeping record of biggest left
    int maxright      {-1};                                                 // keeping record of biggest right
    int finaloutput {0};                                                    // storing final output

    // building left-max
    for(int i = 1; i<height.size(); ++i){

        // calculating indices
        forwardindex   = i;                                                 // forward-index
        backwardindex  = height.size()-1-i;                                 // backward-index

        // calculating maxleft
        maxleft     = height[forwardindex-1] > maxleft ?
                      height[forwardindex-1] : maxleft;                     // running max-left
        leftmaxes[forwardindex] = maxleft;                                  // storing to vector

        // calculating max right
        maxright    = height[backwardindex+1] > maxright ?
                      height[backwardindex+1] : maxright;                   // running max-right
        rightmaxes[backwardindex] = maxright;                               // storing to vector
    }

    // going through the array to calculate maxvolume held by each column
```

```
    for(int i = 0; i < height.size(); ++i){

        // finding max-height of the current column
        auto minheight    = std::min({leftmaxes[i], rightmaxes[i]});     // finding max-height of borders
        auto columnheight = minheight - height[i];                       // subtracting to find space
        columnheight      = columnheight > 0 ? columnheight : 0;         // in case curr-height > max-height
        finaloutput       += columnheight;                               // accumlating to water content
    }

    // printing the final output
    cout << format("finaloutput = {}\n", finaloutput);

    // return
    return(0);

}
```

45 Jump Game II

You are given a 0-indexed array of integers nums of length n. You are initially positioned at index 0. Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at index i, you can jump to any index (i + j) where:

- $0 \leq \mathrm{j} \leq \mathrm{nums[i]}$
- $i + j \leq n$

Return the minimum number of jumps to reach index n - 1. The test cases are generated such that you can reach index n - 1.

Examples

1. **Example 1**
   - Input: nums = [2,3,1,1,4]
   - Output: 2
   - Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.
2. **Example 2**
   - Input: nums = [2,3,0,1,4]
   - Output: 2

Constraints

- $1 \le \text{nums.length} \le 10^4$
- $0 \le \text{nums[i]} \le 1000$
- It's guaranteed that you can reach nums[n - 1].

Code

```cpp
int main(){

    // input- configuration
    vector<int> nums  {2,3,0,1,4};

    // setup
    Timer timer;                                                // setting a timer
    vector<int> minjumps(nums.size(),0);                        // the dp table
    int leftboundary  {-1};                                     // variable to hold the left-boundary
    int rightboundary {-1};                                     // variable to hold the right-boundary

    // moving from the back
    for(int i = nums.size()-2; i>=0; --i){

        // continuign if nums[i] = 0
        if (nums[i] == 0) {
            minjumps[i] = std::numeric_limits<int>::max();      // to prevent this from being chosen
            continue;                                           // moving to next index
        }

        // range of values it can go from here
        leftboundary  = i+1;                                    // the starting point of range
        rightboundary = i+nums[i];                              // the end point of range
```

```
        rightboundary = rightboundary < nums.size()-1 ?
                        rightboundary : nums.size()-1;                    // ensuring within vector range

        // calculating smallest element in range
        auto it = std::min_element(minjumps.begin()+leftboundary,
                                   minjumps.begin()+rightboundary+1);     // finding the minimum value in the range

        // addding min-element to the array
        if (*it == std::numeric_limits<int>::max())
            minjumps[i] = std::numeric_limits<int>::max();                // ensuring infty logic
        else
            minjumps[i] = (1 + *it);                                      // for regular values

    }

    // printing
    cout << format("finaloutput = {}\n", minjumps[0]);
    timer.measure();

    // return
    return(0);

}
```

48. Rotate Image

You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

Examples

1. **Example 1**
   - Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
   - Output: [[7,4,1],[8,5,2],[9,6,3]]
2. **Example 2**
   - Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
   - Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

Constraints

- n == matrix.length == matrix[i].length
- $1 \leq n \leq 20$
- $-1000 \leq matrix[i][j] \leq 1000$

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    vector<vector<int>> matrix {
        {5,1,9,11},
        {2,4,8,10},
        {13,3,6,7},
        {15,14,12,16}
    };


    // setup
    int t_edge = 0;
    int b_edge = matrix.size()-1;
    int l_edge = 0;
    int r_edge = matrix[0].size() -1;
    int temp;

    // shifting layer by layer.
    int numlayers = (matrix.size())/2;

    // shifting layer by layer
    for(int i = 0; i<numlayers; ++i){

        // breaking if they're the same for some reason
        if(t_edge == b_edge || l_edge == r_edge) break;

        // calculatin width
        int currwidth = r_edge - l_edge + 1;
        for(int j = 0; j<currwidth-1; ++j){
```

```cpp

            // shifting the four elements for each
            temp                      = matrix[t_edge][l_edge+j];
            matrix[t_edge][l_edge+j]  = matrix[b_edge-j][l_edge];
            matrix[b_edge-j][l_edge]  = matrix[b_edge][r_edge-j];
            matrix[b_edge][r_edge-j]  = matrix[t_edge+j][r_edge];
            matrix[t_edge+j][r_edge]    = temp;
        }

        // updating edge-parameters based on the layer we're working with
        t_edge += 1;
        b_edge -= 1;
        l_edge += 1;
        r_edge -= 1;

    }

    // printing the matrix
    cout << "final-matrix = \n";
    for(const auto& x: matrix){
        for(const auto& y: x){
            cout << format("{}, ", y);
        }
        cout << format("\n");
    }

    // return
    return(0);

}
```

49. Group Anagrams

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

Examples

1. **Example 1:**
   - Input: strs = [”eat”,”tea”,”tan”,”ate”,”nat”,”bat”]
   - Output: [[”bat”],[”nat”,”tan”],[”ate”,”eat”,”tea”]]
   - Explanation:
     - There is no string in strs that can be rearranged to form ”bat”.
     - The strings ”nat” and ”tan” are anagrams as they can be rearranged to form each other.
     - The strings ”ate”, ”eat”, and ”tea” are anagrams as they can be rearranged to form each other.
2. **Example 2:**
   - Input: strs = [””]
   - Output: [[””]]
3. **Example 3:**
   - Input: strs = [”a”]
   - Output: [[”a”]]

Constraints

- $1 \le \text{strs.length} \le 10^4$
- $0 \le \text{strs[i].length} \le 100$
- strs[i] consists of lowercase English letters.

Code

```cpp
// build histogram
vector<int> fBuildHist(string input){
    vector<int> myhist(26, 0);
    for(auto x: input) {++myhist[static_cast<int>(x - 'a')];}
    return myhist;
}

// Custom hash function for vector<int>
struct VectorHash {
    std::size_t operator()(const std::vector<int>& v) const {
        std::size_t hashValue = 0;
        for (int num : v)
            hashValue ^= std::hash<int>{}(num) + 0x9e3779b9 + (hashValue << 6) + (hashValue >> 2);

        return hashValue;
    }
};
// int main
int main(){

    // starting timer
    Timer timer;

```

```
    // input- configuration
    vector<string> strs{
        "eat",
        "tea",
        "tan",
        "ate",
        "nat",
        "bat"
    };

    // unordered map for hist to vector of vector of strings
    unordered_map< vector<int>, vector<string>, VectorHash > histToGroup;

    for(auto x: strs){

        // building hist of current word
        auto xhist = fBuildHist(x);

        // checking if it exists
        if (histToGroup.find(xhist) == histToGroup.end())
            histToGroup[xhist] = vector<string>({x});
        else
            histToGroup[xhist].push_back(x);

    }

    // building final output
    vector<vector<string>> finalOutput;
    for(auto x: histToGroup) {finalOutput.push_back(x.second);}

    // printing
    cout << format("final-output = {}\n", finalOutput);

    // return
    return(0);
```

```
}
```

54. Spiral Matrix

Given an m x n matrix, return all elements of the matrix in spiral order.

Examples

1. **Example 1:**
    - Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
    - Output: [1,2,3,6,9,8,7,4,5]
2. **Example 2:**
    - Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
    - Output: [1,2,3,4,8,12,11,10,9,5,6,7]

Constraints

- m == matrix.length
- n == matrix[i].length
- $1 \leq m, n \leq 10$
- $-100 \leq matrix[i][j] \leq 100$

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    vector<vector<int>> matrix{
        {1,2,3,4},
        {5,6,7,8},
        {9,10,11,12}
    };

    // setup
    int         left   {0};
    int         right  {static_cast<int>(matrix[0].size()-1)};
    int         top    {0};
    int         bottom {static_cast<int>(matrix.size()-1)};
    vector<int> finalOutput;

    // moving through this
    while(left <= right && top <= bottom){

        // moving on upside
        for(int i = left; i <= right; ++i) {finalOutput.push_back(matrix[top][i]);}

        // moving through the right side
        for(int i = top+1; i<=bottom-1; ++i) {finalOutput.push_back(matrix[i][right]);}

        // moving through the bottom
        for(int i = right; top != bottom && i>= left; --i) {finalOutput.push_back(matrix[bottom][i]);}

        // moving through the left
        for(int i = bottom-1; left!=right && i>= top+1; --i) {finalOutput.push_back(matrix[i][left]);}
```

```
        // updating boundaries
        ++left; --right; ++top; --bottom;

    }

    // printing the finaloutput
    cout << format("final-output = {}\n", finalOutput);

    // return
    return(0);

}
```

55. Jump Game

You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position. Return true if you can reach the last index, or false otherwise.

Examples

1. **Example 1**

   - Input: nums = [2,3,1,1,4]
   - Output: true
   - Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

2. **Example 2**

   - Input: nums = [3,2,1,0,4]
   - Output: false
   - Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

Constraints

- $1 \leq \text{nums.length} \leq 10^4$
- $0 \leq \text{nums[i]} \leq 10^5$

Code

```
int main(){

    // input- configuration
    vector<int> nums  {3,2,1,0,4};

    // setup
    Timer timer;                                                        // starting a timer
    int maxjumpdistance   {0};                                          // variable holding max-jump-distance
    int currjumpdistance  {0};                                          // variable holding max-jump-distance from here
    int finaloutput       {0};                                          // variable holding final verdict

    // going through the nums
    for(int i = 0; i<=maxjumpdistance && i<nums.size(); ++i){

        // calculating max-distance we can go from here
        currjumpdistance  = i + nums[i];

        // updating max-jumpdistance
        maxjumpdistance = currjumpdistance > maxjumpdistance ? \
                          currjumpdistance : maxjumpdistance;

    }

    // updating the final output
    finaloutput = maxjumpdistance >= nums.size()-1 ? true : false;

    // printing the thing
    cout << format("final-output = {}\n", finaloutput);
    timer.measure();


    // return
    return(0);
```

```
}
```

56. Merge Intervals

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

Examples

1. **Example 1:**
    - Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
    - Output: [[1,6],[8,10],[15,18]]
    - Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
2. **Example 2:**
    - Input: intervals = [[1,4],[4,5]]
    - Output: [[1,5]]
    - Explanation: Intervals [1,4] and [4,5] are considered overlapping.

Constraints

- $1 \leq \text{intervals.length} \leq 104$
- $\text{intervals[i].length} == 2$
- $0 \leq \text{starti} \leq \text{endi} \leq 104$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    vector<vector<int>> intervals{
        {1,3},
        {2,6},
        {8,10},
        {15,18}
    };

    // setup
    vector<vector<int>> finalOutput;

    // sorting the input
    std::sort(intervals.begin(), intervals.end(),
        [](const vector<int>& a, const vector<int>& b){return a[0] < b[0];});

    // going through the intervals
    vector<int> runninginterval {intervals[0][0], intervals[0][1]};
    for(int i = 1; i<intervals.size(); ++i){
        if (runninginterval[1] < intervals[i][0]){
            finalOutput.push_back(runninginterval);                         // pushing the results
            runninginterval = intervals[i];                                 // getting new running interfval
        }
        else    {runninginterval[1] = max(runninginterval[1], intervals[i][1]);}
    }
    finalOutput.push_back(runninginterval);                                 // pushing the results

    // returning
    cout << format("final-output = {}\n", finalOutput);
```

```

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

57. Insert Interval

You are given an array of non-overlapping intervals intervals where intervals[i] = [starti, endi] represent the start and the end of the ith interval and intervals is sorted in ascending order by starti. You are also given an interval newInterval = [start, end] that represents the start and end of another interval.

Insert newInterval into intervals such that intervals is still sorted in ascending order by starti and intervals still does not have any overlapping intervals (merge overlapping intervals if necessary).

Return intervals after the insertion.

Note that you don't need to modify intervals in-place. You can make a new array and return it.

Examples

1. **Example 1:**
    - Input: intervals = [[1,3],[6,9]], newInterval = [2,5]
    - Output: [[1,5],[6,9]]
2. **Example 2:**
    - Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
    - Output: [[1,2],[3,10],[12,16]]
    - Explanation: Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].

Constraints

- $0 \leq \text{intervals.length} \leq 10^4$

- intervals[i].length == 2
- $0 \leq$ starti $\leq$ endi $\leq 10^5$
- intervals is sorted by starti in ascending order.
- newInterval.length == 2
- $0 \leq$ start$\leq$ end $\leq 10^5$

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    vector<vector<int>> intervals{
        {1,2},
        {3,5},
        {6,7},
        {8,10},
        {12,16}
    };
    vector<int> newInterval{4,8};


    // setup
    vector<vector<int>> finalOutput;
    intervals.push_back(newInterval);

    // sorting the interval
```

```cpp
    std::sort(intervals.begin(),
              intervals.end(),
              [](const vector<int>& a,
                 const vector<int>& b){return a[0] < b[0];});
    vector<int>       runninginterval {intervals[0]};

    // going through the inputs
    for(int i = 1; i<intervals.size(); ++i){
        if (runninginterval[1] < intervals[i][0]){
            finalOutput.push_back(runninginterval);
            runninginterval = intervals[i];
        }
        else{
            runninginterval[1] = max(runninginterval[1], intervals[i][1]);
        }
    }

    // pushing it back
    finalOutput.push_back(runninginterval);
    cout << format("final-output = {}\n", finalOutput);

    // return
    return(0);

}
```

58. Length of Last Word

Given a string s consisting of words and spaces, return the length of the last word in the string. A word is a maximal substring consisting of non-space characters only.

Example

1. **Example 1:**
    - Input: s = ”Hello World”
    - Output: 5
    - Explanation: The last word is ”World” with length 5.
2. **Example 2:**
    - Input: s = ” fly me to the moon ”
    - Output: 4
    - Explanation: The last word is ”moon” with length 4.
3. **Example 3:**
    - Input: s = ”luffy is still joyboy”
    - Output: 6
    - Explanation: The last word is ”joyboy” with length 6.

Constraints

- $1 \leq \text{s.length} \leq 10^4$
- s consists of only English letters and spaces ' '.
- There will be at least one word in s.

Code

```
int main(){

    // input- configuration
    string s   {"  fly me  to   the moon "};

    // setup
    int p1          {-1};
    int finaloutput {-1};
    string laststring;

    // moving from the end
    for(int i = s.size()-1; i>=0; --i){

        // continuing until you find a non-space character
        if (s[i] == ' ') {continue;}

        // launch the start of first word
        p1 = i;

        // moving p1 until we find the first space or nonword thing
        while(p1>=0 && s[p1]!=' ') {--p1;}

        // calculating the length
```

```
        finaloutput = i - p1;
        laststring = string(s.begin() + p1, s.begin() + i+1);

        // breaking
        break;
    }

    // printing
    cout << format("length = {}, last-word = {}\n", finaloutput, laststring);

    // return
    return(0);

}
```

61. Rotate List

Given the head of a linked list, rotate the list to the right by k places.

Examples

1. **Example 1:**
    - Input: head = [1,2,3,4,5], k = 2
    - Output: [4,5,1,2,3]
2. **Example 2:**
    - Input: head = [0,1,2], k = 4
    - Output: [2,0,1]

Constraints

- The number of nodes in the list is in the range [0, 500].
- -100 $\leq$ Node.val $\leq$ 100
- $0 \leq k \leq 2 * 10^9$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto head  {new ListNode(1)};
    head->next = new ListNode(2);
    head->next->next = new ListNode(3);
    head->next->next->next = new ListNode(4);
    head->next->next->next->next = new ListNode(5);
    auto k {2};

    // trivial cases
    if (head == nullptr) {fPrintLinkedList("head = ", head); return 0;}

    // storing the starting point
    auto startingpoint {head};
    auto listsize    {0};

    // connecting the last-node with the first
    auto traveller {head};
    while(traveller->next) {traveller = traveller->next; ++listsize;}
    ++listsize;
    traveller->next = head;

    // to get rid of cycles
    k = k % listsize;
    auto cutoffpoint  {listsize - k};
    auto count        {0};
    auto finalresult  {static_cast<ListNode*>(nullptr)};

    // finding the cutting off point
```

```
    traveller = head;
    while(traveller){
        ++count;                                                        // appending count
        if (count == cutoffpoint){
            finalresult = traveller->next;                              // setting up the new head
            traveller->next = nullptr;                                  // setting up last pointer
            break;                                                      // breaking up
        }
        traveller = traveller->next;                                    // moving to the next point
    }

    // printing the linked list
    fPrintLinkedList("final-result = ", finalresult);

    // return
    return(0);

}
```

68. Text Justification

Given an array of strings words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left-justified, and no extra space is inserted between words.

Note:

1. A word is defined as a character sequence consisting of non-space characters only.
2. Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.
3. The input array words contains at least one word.

Examples

1. **Example 1:**
    - Input: words = [”This”, ”is”, ”an”, ”example”, ”of”, ”text”, ”justification.”], maxWidth = 16
    - Output:
        - [
        - ”This is an”,
        - ”example of text”,

- ”justification. ”
-]

2. **Example 2:**

   - Input: words = [”What”,”must”,”be”,”acknowledgment”,”shall”,”be”], maxWidth = 16
   - Output:
     - [
     - ”What must be”,
     - ”acknowledgment ”,
     - ”shall be ”
     -]
   - Explanation: Note that the last line is ”shall be ” instead of ”shall be”, because the last line must be left-justified instead of fully-justified. Note that the second line is also left-justified because it contains only one word.

3. **Example 3:**

   - Input: words = [”Science”,”is”,”what”,”we”,”understand”,”well”,”enough”,”to”,”explain”,”to”,”a”,”computer.”,”Art”,”is”,”everythin
     maxWidth = 20
   - Output:
     - [
     - ”Science is what we”,
     - ”understand well”,
     - ”enough to explain to”,
     - ”a computer. Art is”,
     - ”everything else we”,
     - ”do ”
     -]

Constraints

- $1 \leq \text{words.length} \leq 300$
- $1 \leq \text{words[i].length} \leq 20$
- words[i] consists of only English letters and symbols.
- $1 \leq \text{maxWidth} \leq 100$
- $\text{words[i].length} \leq \text{maxWidth}$

Code

```cpp
// function to calculate number of non-space characters
int fCalcLengthOfTempWithoutSpaces(std::vector<std::string> temp){
    int num_nonspaces = 0;
    for(auto x: temp) num_nonspaces += x.size();
    return num_nonspaces;
}

// function to calculate words formed with temp
int fCalcLengthOfTempWithSpaces(std::vector<std::string> temp){
    int num_nonspaces = 0;
    for(auto x: temp) num_nonspaces += 1+ x.size();
    return num_nonspaces-1;
}

// printing temp
void fPrintTemp(std::vector<std::string> temp){
    // printing temp
    std::cout << "temp = ";
    for(auto x: temp) std::cout << x << ",";
```

```
    std::cout << std::endl;
}

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

    // input- configuration
    auto words     = vector<string>({"This", "is", "an", "example", "of", "text", "justification."});
    auto maxWidth  {16};

        // setup
    std::vector<std::string> finalOutput;

    // going through strings
    int acc = 0;
    int numwords = 0;
    int currwidth = 0;
    std::vector<std::string> temp;

    for(int i = 0; i<words.size(); ++i){

        // updating temp
        temp.push_back(words[i]);    // updating words in temp

        // checking if width has been crossed
        if (fCalcLengthOfTempWithSpaces(temp) >= maxWidth || i == words.size()-1){

            // condition temp based on length
            if(fCalcLengthOfTempWithSpaces(temp)>maxWidth){
                temp.pop_back();      // last words gotta go
                --i;                  // making sure its taken care of in next iteration
            }

            // finding length of characters in temp
            int num_nonspaces = fCalcLengthOfTempWithoutSpaces(temp);
```

```cpp

        // finding number of spaces to add
        int numspacestofill = maxWidth - num_nonspaces;

        // calculating numspots
        int numspots = temp.size()-1;

        // calculating ideal number of spaces
        int idealnumspacesperspot;
        int remainders;
        if (numspots!=0){
            idealnumspacesperspot = numspacestofill/numspots;
            remainders            = numspacestofill%numspots;
        }
        else{
            idealnumspacesperspot = numspacestofill/1;
            remainders            = numspacestofill%1;
        }

        // constructing candidate string
        std::string candidate;

        // adding each word in temp to the candidate
        for(int j = 0; j < temp.size(); ++j){

            // fetching word
            auto x = temp[j];

            // adding word to candidate
            candidate += x;

            // adding spaces
            if (j!=temp.size()-1)
                for(int var00 = 0; var00 < idealnumspacesperspot; ++var00)
                    candidate += " ";
```

```
                // checking if there is any remainder left
                if (remainders > 0){
                    candidate += " ";                                        // adding another space
                    --remainders;                                            // subtracting
                }
            }

            // checking if there are remaindeers
            while (remainders > 0){
                candidate += " ";                                            // adding another space
                --remainders;                                                // subtracting
            }

            while (candidate.size()!=maxWidth) {candidate += " ";}           // adding another space

            // appending candidate to final output
            finalOutput.push_back(candidate);

            // getting rid of everything
            if (i != words.size()-1) {temp.clear();}
        }
    }

    // making function left justified
    std::string lastline;
    for(int i = 0; i<temp.size(); ++i){
        lastline += temp[i];
        if (i!=temp.size()-1) {lastline += " ";}
    }

    // adding spaces until end
    while(lastline.size()!=maxWidth) {lastline += " ";}

    // replacing last line
```

```
    finalOutput[finalOutput.size()-1] = lastline;

    // return
    return(0);

}
```

71. Simplify Path

You are given an absolute path for a Unix-style file system, which always begins with a slash '/'. Your task is to transform this absolute path into its simplified canonical path.

The rules of a Unix-style file system are as follows:

- A single period '.' represents the current directory.
- A double period '..' represents the previous/parent directory.
- Multiple consecutive slashes such as '//' and '///' are treated as a single slash '/'.
- Any sequence of periods that does not match the rules above should be treated as a valid directory or file name. For example, '...' and '....' are valid directory or file names.

The simplified canonical path should follow these rules:

- The path must start with a single slash '/'.
- Directories within the path must be separated by exactly one slash '/'.
- The path must not end with a slash '/', unless it is the root directory.
- The path must not have any single or double periods ('.' and '..') used to denote current or parent directories.

Return the simplified canonical path.

Examples

1. **Example 1:**
   - Input: path = "/home/"
   - Output: "/home"
   - Explanation: The trailing slash should be removed.
2. **Example 2:**
   - Input: path = "/home//foo/"
   - Output: "/home/foo"
   - Explanation: Multiple consecutive slashes are replaced by a single one.
3. **Example 3:**
   - Input: path = "/home/user/Documents/../Pictures"
   - Output: "/home/user/Pictures"
   - Explanation: A double period ".." refers to the directory up a level (the parent directory).

Constraints

- $1 \leq \text{path.length} \leq 3000$
- path consists of English letters, digits, period '.', slash '/' or '_'.
- path is a valid absolute Unix path.

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    string path    {"/home//foo/"};

    // setup
    vector<string> stack;

    // going through the array
    int i = 0 ;
    while (i < path.size()){

        // moving until end of string or a /
        string temp;
        while(path[i]!='/' && i<path.size()){
            temp += std::string(1,path[i++]);
        }

        // continuing if temp size is zero
        if (temp.size()==0)   {++i; continue;}

        // evaluating the current string
        if (temp == ".")      {++i; continue;}
        else if (temp == "..") {
            if (stack.size()!=0) {stack.pop_back();}
            ++i;}
        else   {stack.push_back(temp); ++i;}
    }


```

```
    // reconstructing final path
    string finalOutput;

    // reconstructing
    for(auto x: stack) {finalOutput += "/" + x;}

    // size thing
    if (finalOutput.size() == 0) {finalOutput += "/";}

    // returning
    cout << format("finalOutput = {}\n", finalOutput);

    // return
    return(0);

}
```

73. Set Matrix Zeroes

Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's. You must do it in place.

Examples

1. **Example 1**
    - Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
    - Output: [[1,0,1],[0,0,0],[1,0,1]]
2. **Example 2**
    - Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
    - Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]

Constraints

- m == matrix.length
- n == matrix[0].length
- $1 \leq$ m, n $\leq$ 200
- -231 $\leq$ matrix[i][j] $\leq$ 231 - 1

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    vector<vector<int>> matrix{
        {0,1,2,0},
        {3,4,5,2},
        {1,3,1,5}
    };

    // setup
    vector<bool> colregister(matrix[0].size(), false); colregister.reserve(colregister.size());
    vector<bool> rowregister(matrix.size(), false); rowregister.reserve(rowregister.size());

    // registering
    #pragma omp parallel for
    for(int i = 0; i<matrix.size(); ++i)
        for(int j = 0; j<matrix[0].size(); ++j)
            if (matrix[i][j] == 0) {colregister[j] = true; rowregister[i] = true;}

    // rewriting
    #pragma omp parallel for
    for(int i = 0; i<matrix.size(); ++i)
        for(int j = 0; j<matrix[0].size(); ++j)
            if (colregister[j] == true || rowregister[i] == true) {matrix[i][j] = 0;}

    // printing the final output
    cout << format("final-output \n");
    for(const auto& x: matrix){
        for(const auto& y: x) {cout << format("{}, ", y);}
        cout << format("\n");
```

```
    }

    // return
    return(0);

}
```

77. Combinations

Given two integers n and k, return all possible combinations of k numbers chosen from the range [1, n].

You may return the answer in any order.

Examples

1. **Example 1:**

    - Input: n = 4, k = 2
    - Output: [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]
    - Explanation: There are 4 choose 2 = 6 total combinations.
        - Note that combinations are unordered, i.e., [1,2] and [2,1] are considered to be the same combination.

2. **Example 2:**

    - Input: n = 1, k = 1
    - Output: [[1]]
    - Explanation: There is 1 choose 1 = 1 total combination.

Constraints

- $1 \leq n \leq 20$
- $1 \leq k \leq n$

Code

```cpp
void foo(vector<int>          pathsofar,
         int                  n,
         int                  k,
         vector<vector<int>>& finalOutput){

    // checking legnth of pathsofar
    if (pathsofar.size() > k) return;

    // adding the current path to the final output if the length is k
    if (pathsofar.size() == k) {finalOutput.push_back(pathsofar); return;}

    // setting up starting point
    int startingpoint;
    if (pathsofar.size() == 0) startingpoint = 0+1;
    else                       startingpoint = pathsofar[pathsofar.size()-1]+1;

    // running through the options
    for(int i = startingpoint; i<=n; ++i){

        // calling the graph on those
        auto pathsofar_temp = pathsofar;
        pathsofar_temp.push_back(i);

        // calling the functiona again
        foo(pathsofar_temp, n, k, finalOutput);
    }

    // returning
    return;
}

int main(){
```

```
    // starting timer
    Timer timer;

    // input- configuration
    auto n {4};
    auto k {2};

    // setup
    vector< vector<int> > finalOutput;
    vector<int>           pathsofar;

    // calling the function
    foo(pathsofar, n, k, finalOutput);

    // printing the output
    cout << format("finalOutput = {}\n", finalOutput);

    // return
    return(0);

}
```

80. Remove Duplicates from Sorted Array II

Given an integer array nums sorted in non-decreasing order, remove some duplicates in-place such that each unique element appears at most twice. The relative order of the elements should be kept the same.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the first part of the array nums. More formally, if there are k elements after removing the duplicates, then the first k elements of nums should hold the final result. It does not matter what you leave beyond the first k elements.

Return k after placing the final result in the first k slots of nums.

Do not allocate extra space for another array. You must do this by modifying the input array in-place with O(1) extra memory.

1. **Example 1**
   - `Input: nums = [1,1,1,2,2,3]`
   - `Output: 5, nums = [1,1,2,2,3,_]`
2. **Example 2**
   - `Input: nums = [0,0,1,1,1,1,2,3,3]`
   - `Output: 7, nums = [0,0,1,1,2,3,3,_,_]`

Code

```
int main(){

    // input- configuration
    vector<int> nums  {1,1,1,2,2,3};

```

```
    // setup
    int destination   {1};
    int prev          {nums[0]};
    int element_counter {1};
    int numwrites     {1};

    // going through the values
    for(int i = 1; i < nums.size(); ++i){

        // updating counter
        if (nums[i-1] == nums[i]) {++element_counter;}
        else                     {element_counter = 1;}

        // checking the element counters
        if (element_counter <=2) {nums[destination++] = nums[i];}

    }

    // printing the final output
    cout << format("nums = "); fpv(nums);
    cout << format("return-value = {}\n", destination);

    // return
    return(0);

}
```

86. Partition List

Given the head of a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

Examples

1. **Example 1:**
    - Input: head = [1,4,3,2,5,2], x = 3
    - Output: [1,2,2,4,3,5]
2. **Example 2:**
    - Input: head = [2,1], x = 2
    - Output: [1,2]

Constraints

- The number of nodes in the list is in the range [0, 200].
- $-100 \leq \text{Node.val} \leq 100$
- $-200 \leq x \leq 200$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto head  {new ListNode(1)};
    head->next = new ListNode(4);
    head->next->next = new ListNode(3);
    head->next->next->next = new ListNode(2);
    head->next->next->next->next = new ListNode(5);
    head->next->next->next->next->next = new ListNode(2);
    auto x {3};

    // setup
    ListNode* traveller      = head;
    ListNode* lefthead       = new ListNode(-1);
    ListNode* lefttraveller  = lefthead;
    ListNode* righthead      = new ListNode(-1);
    ListNode* righttraveller = righthead;

    // going through the list
    while(traveller){
        // checking next-value
        if (traveller->val < x){
            lefttraveller->next   = new ListNode(traveller->val);
            lefttraveller         = lefttraveller->next;
        }
        else {
            righttraveller->next  = new ListNode(traveller->val);
            righttraveller        = righttraveller->next;
        }

```

```
        // moving traveller
        traveller = traveller->next;
    }

    // connecting
    lefttraveller->next = righthead->next;

    // returning
    fPrintLinkedList("Final-output = ", lefthead->next);

    // 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.

Examples

1. **Example 1:**
   - Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
   - Output: [1,2,2,3,5,6]
   - Explanation: The arrays we are merging are [1,2,3] and [2,5,6]. The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.
2. **Example 2:**
   - Input: nums1 = [1], m = 1, nums2 = [], n = 0
   - Output: [1]
   - Explanation: The arrays we are merging are [1] and []. The result of the merge is [1].
3. **Example 3:**
   - Input: nums1 = [0], m = 0, nums2 = [1], n = 1

- Output: [1]
- Explanation: The arrays we are merging are [] and [1]. The result of the merge is [1]. Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the merge result can fit in nums1.

Constraints:

1. nums1.length == m + n
2. nums2.length == n
3. $0 \leq$ m, n ≤ 200
4. $1 \leq$ m + n ≤ 200
5. $-10^9 \leq$ nums1[i], nums2[j] $\leq 10^9$

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);
}
```

92. Reverse Linked List II

Given the head of a singly linked list and two integers left and right where left ¡= right, reverse the nodes of the list from position left to position right, and return the reversed list.

Examples

1. **Example 1:**
    - Input: head = [1,2,3,4,5], left = 2, right = 4
    - Output: [1,4,3,2,5]
2. **Example 2:**
    - Input: head = [5], left = 1, right = 1
    - Output: [5]

Constraints

- The number of nodes in the list is n.
- $1 \leq n \leq 500$
- $-500 \leq \text{Node.val} \leq 500$
- $1 \leq \text{left} \leq \text{right} \leq n$

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    ListNode* head                    = new ListNode(1);
    head->next                        = new ListNode(2);
    head->next->next                  = new ListNode(3);
    head->next->next->next            = new ListNode(4);
    head->next->next->next->next = new ListNode(5);
    auto left  {2};
    auto right {4};

    // trivial case
    if (left == right) {fPrintLinkedList(head); cout << endl; return 0;}

    // setup
    ListNode* traveller   = nullptr;
    ListNode* entrypoint  = nullptr;
    ListNode* exitpoint   = nullptr;
    ListNode* revhead     = nullptr;
    ListNode* dummy       = new ListNode(0);
    dummy->next           = head;

    // moving through list
    int nodeposition = -1;
    traveller = dummy;
    while(traveller){

        ++nodeposition;   // updating node position

        // finding important points
```

```
        if(nodeposition == left-1)       {entrypoint = traveller;}
        else if(nodeposition == left)    {revhead   = traveller;}
        else if(nodeposition == right+1) {exitpoint = traveller;}

        // moving traveller to next point
        traveller = traveller->next;
    }

    // reversing the head
    ListNode* prev        = nullptr;
    ListNode* actualright = nullptr;
    traveller             = revhead;
    nodeposition          = left;

    while(traveller && nodeposition <= right){
        ++nodeposition;
        actualright    = traveller->next; // storing original right
        traveller->next = prev;          // reconnection
        prev           = traveller;      // storing prev for next iteration
        traveller      = actualright;    // moving to original right
    }

    // tying things together
    entrypoint->next  = prev;
    revhead->next     = exitpoint;

    // returning dummy-es nstex
    fPrintLinkedList(dummy->next); cout << endl;

    // return
    return(0);

}
```

98. Validate Binary Search Tree

Given the root of a binary tree, determine if it is a valid binary search tree (BST).

A valid BST is defined as follows:

- The left subtree of a node contains only nodes with keys strictly less than the node's key.
- The right subtree of a node contains only nodes with keys strictly greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

Constraints

- The number of nodes in the tree is in the range [1, 104].
- $-2^{31} \leq \text{Node.val} \leq 2^{31} - 1$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root       {new TreeNode(2)};
    root->left      = new TreeNode(1);
    root->right     = new TreeNode(3);

```

```
    // setup
    bool validBSTFlag = true;
    vector<int> nodevalues;

    std::function<void(const TreeNode*)> foo = [&foo, &nodevalues, &validBSTFlag](
        const TreeNode* root){

        // if false, stop everything
        if (validBSTFlag == false)   {return;}

        // returning
        if (root == nullptr)         {return;}

        // going left
        if (root->left)              {foo(root->left);}

        // adding current
        if (nodevalues.size() == 0)  {nodevalues.push_back(root->val);}
        else{
            // check top
            auto topvalue = nodevalues[nodevalues.size()-1];

            // check if top value is less than or equal
            if (topvalue >= root->val) {validBSTFlag = false;}            // since this is not valid
            else                     {nodevalues.push_back(root->val);}
        }

        // going right
        if (root->right)             {foo(root->right);}
    };

    // calling function
    foo(root);

    // returning
```

```
    cout << format("final-output = {}\n", validBSTFlag);

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

100. Same Tree

Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Examples

1. **Example 1:**
   - Input: p = [1,2,3], q = [1,2,3]
   - Output: true
2. **Example 2:**
   - Input: p = [1,2], q = [1,null,2]
   - Output: false
3. **Example 3:**
   - Input: p = [1,2,1], q = [1,1,2]
   - Output: false

Constraints

- The number of nodes in both trees is in the range [0, 100].
- $-10^4 \leq \text{Node.val} \leq 10^4$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto p {new TreeNode(1)};
    p->left    = new TreeNode(2);
    p->right   = new TreeNode(3);

    auto q {new TreeNode(1)};
    q->left    = new TreeNode(2);
    q->right   = new TreeNode(3);

    // setup
    std::function<bool(TreeNode*, TreeNode*)> fCompareTrees = [&fCompareTrees](
        TreeNode* p, TreeNode* q){

        // basecase
        if (p == nullptr && q == nullptr)   return true;
        if (p == nullptr && q != nullptr)   return false;
        if (p != nullptr && q == nullptr)   return false;
        if (p->val != q->val)               return false;

        // going through the next set of branches
        return (fCompareTrees(p->left, q->left) && fCompareTrees(p->right, q->right));
    };

    // going through the two nodes
    bool finalOutput = fCompareTrees(p, q);

    // returning final output
    cout << format("final-output = {}\n", finalOutput);
```

```

    // return
    return(0);

}
```

101. Symmetric Tree

Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

Examples

1. **Example 1:**
    - Input: root = [1,2,2,3,4,4,3]
    - Output: true
2. **Example 2:**
    - Input: root = [1,2,2,null,3,null,3]
    - Output: false

Constraints

- The number of nodes in the tree is in the range [1, 1000].
- -100 $\leq$ Node.val $\leq$ 100

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root         {new TreeNode(1)};
    root->left         = new TreeNode(2);
    root->right        = new TreeNode(2);
    root->left->left   = new TreeNode(3);
    root->left->right  = new TreeNode(4);
    root->right->left  = new TreeNode(4);
    root->right->right = new TreeNode(3);

    // setup
    std::function<bool(TreeNode*, TreeNode*)> dfs = [&dfs](
        TreeNode* leftNode,
        TreeNode* rightNode){

        // printing the base-cases
        if (leftNode == nullptr && rightNode == nullptr) return true;
        if (leftNode == nullptr || rightNode == nullptr) return false;

        // calling the children
        bool finaloutput = (leftNode->val == rightNode->val)      && \
                           dfs(leftNode->left, rightNode->right)   && \
                           dfs(leftNode->right, rightNode->left);

        // returning the value
        return finaloutput;
    };

    // running
    cout << format("final-output = {}\n", dfs(root->left, root->right));
```

```

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

102. Binary Tree Level Order Traversal

Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

Constraints

- The number of nodes in the tree is in the range [0, 2000].
- -1000 $\leq$ Node.val $\leq$ 1000

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root               {new TreeNode(3)};
    root->left              = new TreeNode(9);
    root->right             = new TreeNode(20);
    root->right->left       = new TreeNode(15);
    root->right->right      = new TreeNode(7);

    // setup
    std::vector< std::vector<int> > finalOutput;
    std::function<void(const TreeNode*, int)> f = [&f, &finalOutput](
        const TreeNode* root,
        int             level){

        // base-case
```

```
        if (root == nullptr) return;

        // finding current-level
        level += 1;

        // increasing size of the thing
        while (finalOutput.size()<level+1)
            finalOutput.push_back(std::vector<int>());

        // appending to value
        if (finalOutput.size() < level+1)
            finalOutput[level] = std::vector<int>(root->val);
        else
            finalOutput[level].push_back(root->val);

        // calling
        f(root->left, level);
        f(root->right, level);

        // returning return;
        return;

    };

    // going through the tree
    f(root, -1);

    // printing the final output
    cout << format("finalOutput = {}\n", finalOutput);

    // return
    return(0);

}
```

108. Convert Sorted Array to Binary Search Tree

Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree.

Constraints

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 \leq \text{nums[i]} \leq 10^4$
- nums is sorted in a strictly increasing order.

Code

```
void foo(TreeNode*       parent,
        int              parentdirection,
        vector<int>&     nums,
        int              left,
        int              right)
{
    // sending it back if right is less than left
    if (left > right) return;

    // finding mid point
    int mid = (left + right)/2;

    // creating mid-node
    TreeNode* midnode = new TreeNode(nums[mid]);

    // connecting current to parent node
```

```
    if (parentdirection == -1)       {parent->right  = midnode;}
    else if (parentdirection == 1)   {parent->left   = midnode;}

    // going left
    foo(midnode, 1, nums, left, mid-1);

    // going right
    foo(midnode, -1, nums, mid+1, right);

    // returning
    return;

}
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    vector<int> nums  {-10,-3,0,5,9};

    // setup
    TreeNode* dummy = new TreeNode(-1);

    // calling the function
    foo(dummy, -1, nums, 0, nums.size()-1);

    // return
    return(0);

}
```

112. Path Sum

Given the root of a binary tree and an integer targetSum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals targetSum.

Constraints

- The number of nodes in the tree is in the range [0, 5000].
- -1000 $\leq$ Node.val $\leq$ 1000
- -1000 $\leq$ targetSum $\leq$ 1000

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root                {new TreeNode(5)};

    root->left               = new TreeNode(4);
    root->right              = new TreeNode(8);

    root->left->left         = new TreeNode(11);
    root->right->left        = new TreeNode(13);
    root->right->right       = new TreeNode(4);

```

```cpp
    root->left->left->left    = new TreeNode(7);
    root->left->left->right   = new TreeNode(2);
    root->right->right->right = new TreeNode(1);

    auto targetSum    {22};

    // setup
    auto runningsum   {0};
    auto binaryflag   {false};
    std::function<void(const TreeNode*, int)> fRunSum = [&fRunSum,
                  &targetSum,
                  &binaryflag](
        const TreeNode* root,
        int         runningsum
        ){

        // sending it back
        if (root       == nullptr)    return;
        if (binaryflag == true)       return;

        // adding current val to running sum
        runningsum += root->val;

        // checking if there are children
        if (root->left || root->right){
            if (root->left)   fRunSum(root->left,   runningsum);
            if (root->right)  fRunSum(root->right,  runningsum);
        }
        else{
            // in this case, we check the running sum
            if (runningsum == targetSum){ binaryflag = true;}
        }

        // returning
        return;
```

```
    };

    // running
    fRunSum(root, runningsum);

    // printing
    cout << format("final-output = {}\n", binaryflag);

    // return
    return(0);

}
```

114. Flatten Binary Tree to Linked List

Given the root of a binary tree, flatten the tree into a “linked list”:

- The “linked list” should use the same TreeNode class where the right child pointer points to the next node in the list and the left child pointer is always null.
- The “linked list” should be in the same order as a pre-order traversal of the binary tree.

Examples

1. **Example 1:**
    - Input: root = [1,2,5,3,4,null,6]
    - Output: [1,null,2,null,3,null,4,null,5,null,6]
2. **Example 2:**
    - Input: root = []
    - Output: []
3. **Example 3:**
    - Input: root = [0]
    - Output: [0]

Constraints

- The number of nodes in the tree is in the range [0, 2000].
- $-100 \leq \text{Node.val} \leq 100$

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root         {new TreeNode(1)};
    root->left         = new TreeNode(2);
    root->right        = new TreeNode(5);
    root->left->left   = new TreeNode(3);
    root->left->right  = new TreeNode(4);
    root->right->right = new TreeNode(6);

    // recursion lambda
    auto foo = [&foo](TreeNode*          root,
                      vector<TreeNode*>&  nodeaddresses){

        // sending it back
        if (root == nullptr) return;

        // adding address
        nodeaddresses.push_back(root);

        // going down left
        if (root->left) foo(root->left, nodeaddresses);
```

```
        if (root->right) foo(root->right, nodeaddresses);

        // returning
        return;
    };

    // trivial case
    if (root == nullptr) return 0;

    // setup
    vector<TreeNode*> nodeaddresses;

    // fill registers
    foo(root, nodeaddresses);

    // going through the addresses and reconnecting it
    for(int i = 0; i<nodeaddresses.size()-1; ++i){
        nodeaddresses[i]->left = nullptr;
        nodeaddresses[i]->right = nodeaddresses[i+1];
    }

    // taking care of the last
    nodeaddresses[nodeaddresses.size()-1]->left = nullptr;
    nodeaddresses[nodeaddresses.size()-1]->right = nullptr;

    // return
    return(0);

}
```

117. Populating Next Right Pointers in Each Node II

Given a binary tree of type

```
struct Node {
    int val;
    Node *left;
    Node *right;
    Node *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL. Initially, all next pointers are set to NULL.

Examples

1. **Example 1:**

   - Input: root = [1,2,3,4,5,null,7]
   - Output: [1,#,2,3,#,4,5,7,#]
   - Explanation: Given the above binary tree (Figure A), your function should populate each next pointer to point to its next right node, just like in Figure B. The serialized output is in level order as connected by the next pointers, with # signifying the end of each level.

2. **Example 2:**

   - Input: root = []
   - Output: []

Constraints

- The number of nodes in the tree is in the range [0, 6000].
- $-100 \leq$ Node.val ≤ 100

Code

```
class Node {
public:
    int val;
    Node* left;
    Node* right;
    Node* next;

    Node() : val(0), left(NULL), right(NULL), next(NULL) {}
    Node(int _val) : val(_val), left(NULL), right(NULL), next(NULL) {}
    Node(int _val, Node* _left, Node* _right, Node* _next)
        : val(_val), left(_left), right(_right), next(_next) {}
};


int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root         {new Node(1)};
    root->left         = new Node(2);
    root->right        = new Node(3);
    root->left->left   = new Node(4);
    root->left->right = new Node(5);
```

```
    root->right->right = new Node(7);

    // running
    if (root == nullptr) {cout << format("done!\n"); return 0;}

    // calling the function
    int count = -1;
    vector< vector<Node*> > addressvectors;

    // calling th efunction
    foo(root, count, addressvectors);

    // building the connections
    for(auto x: addressvectors){
        int i = 0;
        for(; i<x.size()-1; ++i)
            if (x[i] != nullptr)  {x[i]->next = x[i+1];}                    // look to the right

        // ensuring that the last pointer points to thr right most
        x[i]->next = nullptr;
    }

    // completion
    cout << format("done!\n");

    // return
    return(0);

}
```

121. 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 ith 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.

Examples

1. **Example 1**
    - Input: prices = [7,1,5,3,6,4]
    - Output: 5
2. **Example 2**
    - Input: prices = [7,6,4,3,1]
    - Output: 0

Constraints:

- $1 \leq \text{prices.length} \leq 10^5$
- $0 \leq \text{prices[i]} \leq 10^4$

Code

```cpp
int main(){

    // input- configuration
    vector<int> prices {7,6,4,3,1};

    // setup
    StopWatch timer;                                        // timer-object
    int p0        {0};                                      // first index-pointer
    int p1        {1};                                      // second index-pointer
    int maxprofit {0};                                      // variable to hold max-profit
    int curr      {-1};                                     // variable to hold current-profit

    // going through array
    while(p1<prices.size()){
        curr       = prices[p1] - prices[p0];               // calculating current profit
        maxprofit  = curr > maxprofit ? curr : maxprofit;  // updating max-profit
        if (curr < 0)  {p0 = p1;}                          // updating p0 if we find lower point
        ++p1;
    }

    // printing the final output
    cout << format("maxprofit = {}\n", maxprofit);
    timer.stop();

    // return
    return(0);

}
```

122. Best Time To Buy And Sell Stock II

You are given an integer array prices where prices[i] is the price of a given stock on the ith day. On each day, you may decide to buy and/or sell the stock. You can only hold at most one share of the stock at any time. However, you can buy it then immediately sell it on the same day. Find and return the maximum profit you can achieve.

Examples

1. **Example 1**
    - Input: prices = [7,1,5,3,6,4]
    - Output: 7
    - Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4. Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3. Total profit is 4 + 3 = 7.
2. **Example 2**
    - Input: prices = [1,2,3,4,5]
    - Output: 4
    - Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4. Total profit is 4.
3. **Example 3**
    - Input: prices = [7,6,4,3,1]
    - Output: 0
    - Explanation: There is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0.

Constraints

- $1 \leq \text{prices.length} \leq 3 * 10^4$
- $0 \leq \text{prices[i]} \leq 10^4$

Code

```cpp
int main(){

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

    // setup
    int p1         {0};                                  // index-pointer to buying
    int p2         {0};                                  // index-pointer to selling
    int accprofit  {0};                                  // variable to accumulate profit
    int currprofit {std::numeric_limits<int>::min()};   // variable to hold curr-profit

    // going through this
    while(p2 < prices.size()){

        currprofit = prices[p2] - prices[p1];            // calculating current profit

        if (currprofit > 0){
            accprofit  += currprofit;                    // accumulating the profit
            p1         = p2++;                           // moving the starting point
            continue;                                    // moving into the next iteration
        }
        else if (currprofit < 0){
            p1         = p2++;                           // moving the starting point
            continue;
        }
```

```

        ++p2;                                               // updating p2
    }

    // printing the max-value
    cout << format("accprofit = {}\n", accprofit);

    // return
    return(0);

}
```

124. Binary Tree Maximum Path Sum

A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root.

The path sum of a path is the sum of the node's values in the path.

Given the root of a binary tree, return the maximum path sum of any non-empty path.

Constraints

- The number of nodes in the tree is in the range $[1, 3 * 10^4]$.
- -1000 $\leq$ Node.val $\leq$ 1000

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root          {new TreeNode(-10)};
    root->left          = new TreeNode(9);
    root->right         = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);


    // setup
```

```cpp
    auto finalresult  {std::numeric_limits<int>::min()};

    // creating lambda
    std::function<int(const TreeNode*)> fFindMaxPathSum = [&fFindMaxPathSum,
                                                         &finalresult](
        const TreeNode* root){

        // checking if leaf-node
        if(root->left  == nullptr && root->right == nullptr){

            // checking the final-result with the leaf-node value
            finalresult = finalresult > root->val ? finalresult : root->val;

            // the potential from leaf-node
            return root->val;
        }
        else{

            // creating potentials-vector
            vector<int> potentials;
            if (root->left)                        {potentials.emplace_back(fFindMaxPathSum(root->left));}
            if (root->right)                       {potentials.emplace_back(fFindMaxPathSum(root->right));}

            // calculating sum of left and right
            auto leftplusright {0};
            for(int i = 0; i<potentials.size(); ++i) {leftplusright += potentials[i];}

            // calculating sum of path curr with either paths
            for(int i = 0; i<potentials.size(); ++i) {potentials[i] += root->val;}

            // adding curr-value alone as a candidate
            potentials.push_back(root->val);

            // sending the maximum-value back
            const auto maxelement = *(std::max_element(potentials.begin(), potentials.end()));
```

```cpp

            // path from left-curr-right
            potentials.push_back(leftplusright + root->val);

            // auto temp = *(std::max_element(potentials.begin(), potentials.end()));
            const auto comparitivemax = *(std::max_element(potentials.begin(), potentials.end()));

            // finalresult = std::max(finalresult, maxelement);
            finalresult = finalresult > comparitivemax ? finalresult : comparitivemax;

            // returning the max-potential from here
            return maxelement;
        }
    };


    // calculating the final-result
    fFindMaxPathSum(root);

    // printing
    cout << format("final-result = {}\n", finalresult);

    // return
    return(0);

}
```

125. Valid Palindrome

A phrase is a palindrome if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers. Given a string s, return true if it is a palindrome, or false otherwise.

Examples

1. **Example 1:**
   - Input: s = ”A man, a plan, a canal: Panama”
   - Output: true
   - Explanation: ”amanaplanacanalpanama” is a palindrome.
2. **Example 2:**
   - Input: s = ”race a car”
   - Output: false
   - Explanation: ”raceacar” is not a palindrome.
3. **Example 3:**
   - Input: s = ” ”
   - Output: true
   - Explanation: s is an empty string ”” after removing non-alphanumeric characters. Since an empty string reads the same forward and backward, it is a palindrome.

Constraints

- $1 \leq \text{s.length} \leq 2 * 10^5$
- s consists only of printable ASCII characters.

Code

```
int main(){

    // input- configuration
    auto s = string("A man, a plan, a canal: Panama");

    // setup
    auto pleft        {0};
    auto pright       {static_cast<int>(s.size())-1};
    auto finaloutput  {true};

    // lambda to check if alphanumeric
    auto isalphanumeric = [](const int& x){
        if (x-65 >= 0 && 90-x >= 0)    {return true;}    // upper-case check
        if (x-97 >= 0 && 122-x >= 0)   {return true;}    // lower-case check
        if (x-48 >= 0 && 57-x >= 0)    {return true;}    // numeric check
        return false;
    };

    // running
    while(pleft < s.size() && pright >= 0){

        // moving pleft until we find the position
        while(pleft < s.size() && isalphanumeric(s[pleft]) == false) {++pleft;}
        while(pright >= 0     && isalphanumeric(s[pright]) == false) {--pright;}

```

```
        // checking bounds
        if (pleft>=pright) {break;}

        // checking if they're the same
        if (std::tolower(s[pleft]) != std::tolower(s[pright])) {finaloutput = false; break;}

        // updating pointers
        ++pleft; --pright;
    }

    // printing
    cout << format("final-output = {}\n", finaloutput);

    // return
    return(0);

}
```

127. Word Ladder

A transformation sequence from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord → s1 → s2 → ... → sk such that:

- Every adjacent pair of words differs by a single letter.
- Every si for $1 \leq i \leq k$ is in wordList. Note that beginWord does not need to be in wordList.
- sk == endWord

Given two words, beginWord and endWord, and a dictionary wordList, return the number of words in the shortest transformation sequence from beginWord to endWord, or 0 if no such sequence exists.

Examples

1. **Example 1:**
   - Input: beginWord = ”hit”, endWord = ”cog”, wordList = [”hot”,”dot”,”dog”,”lot”,”log”,”cog”]
   - Output: 5
   - Explanation: One shortest transformation sequence is ”hit” → ”hot” → ”dot” → ”dog” → cog”, which is 5 words long.
2. **Example 2:**
   - Input: beginWord = ”hit”, endWord = ”cog”, wordList = [”hot”,”dot”,”dog”,”lot”,”log”]
   - Output: 0
   - Explanation: The endWord ”cog” is not in wordList, therefore there is no valid transformation sequence.

Constraints

- $1 \leq$ beginWord.length ≤ 10
- endWord.length $==$ beginWord.length
- $1 \leq$ wordList.length ≤ 5000
- wordList[i].length $==$ beginWord.length
- beginWord, endWord, and wordList[i] consist of lowercase English letters.
- beginWord $\neq$ endWord
- All the words in wordList are unique.

Code

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

    // starting timer
    Timer timer;

    // input- configuration
    string beginWord          {"hit"};
    string endWord            {"cog"};
    vector<string> wordList   {
        "hot","dot","dog","lot","log"
    };

    // returning error if endword is not in wordhlist
    if (std::find(wordList.begin(), wordList.end(), endWord) == wordList.end()) {
```

```
        cout << format("final-output = 0\n");
        return 0;
    }

    // setup
    unordered_map<string, std::set<string> > neighbours;

    auto calculatedistances = [](const string a, const string b){
        auto numdiffs  {0};
        for(int i = 0; i<a.size(); ++i) {if (a[i] != b[i]) {++numdiffs;}}
        if (numdiffs == 0)           {return 0;}
        else if (numdiffs == 1)      {return 1;}
        else                         {return std::numeric_limits<int>::max();}
    };

    // starting again
    vector<string> candidates {beginWord};
    for(const auto x: wordList) {if (x!=endWord) candidates.push_back(x);}
    vector<string> nextgencandidates {};

    vector<string> recruiters        {{endWord}};
    vector<string> nextgenrecruiters {};

    // going through the loop
    auto count  {1};
    auto runningcondition {true};
    auto foundpath        {false};

    while(runningcondition){
        // increasing count
        ++count;

        // comparing distance between candidates and recruiters
        for(const auto candidate: candidates){
```

```
            auto shortestpath {std::numeric_limits<int>::max()};
            for(const auto& recruiter: recruiters)
                shortestpath  = std::min(shortestpath, calculatedistances(candidate, recruiter));

            // adding to next-generation of recruiters if diff == 1
            if (shortestpath == 1){
                nextgenrecruiters.push_back(candidate);    // adding to next gen
                if (candidate == beginWord) {foundpath = true; runningcondition = false;}
            }
            else if (shortestpath == std::numeric_limits<int>::max()){
                nextgencandidates.push_back(candidate);
            }
        }

        // rewriting history
        int prevnumcandidates = candidates.size();
        int nexnumcandidates  = nextgencandidates.size();
        candidates = nextgencandidates; nextgencandidates.clear();
        recruiters = nextgenrecruiters; nextgenrecruiters.clear();

        // evaluating running condition
        if (prevnumcandidates == nexnumcandidates) {runningcondition = false;}

    }

    // setting up final output
    if (!foundpath) {count = 0;}

    // printing final-output
    cout << format("final-output = {}\n", count);


    // return
    return(0);

```

```
}
```

128. Longest Consecutive Sequence

Given an unsorted array of integers nums, return the length of the longest consecutive elements sequence.

You must write an algorithm that runs in O(n) time.

Examples

1. **Example 1:**
    - Input: nums = [100,4,200,1,3,2]
    - Output: 4
    - Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.
2. **Example 2:**
    - Input: nums = [0,3,7,2,5,8,4,6,0,1]
    - Output: 9
3. **Example 3:**
    - Input: nums = [1,0,1,2]
    - Output: 3

Constraints

- $0 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums[i]} \leq 10^9$

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto nums = vector<int>({0,3,7,2,5,8,4,6,0,1});

    // setup
    std::multiset<int> var00;

    // trivial-cases
    if (nums.size() <= 1) {cout << format("final-output = {}\n", nums.size()); return 0;}

    // going through elements
    for(auto x: nums) {var00.insert(x);}

    // going through elements
    int maxlength = std::numeric_limits<int>::min();
    int temp = 1;
    std::deque<int> var01;

    for(auto x: var00){

        if (var01.size()<1)          {var01.push_back(x);}
        else{
            // comparing previous element and current
            if (x - var01[0] == 1)   {++temp; var01[0] = x; maxlength = max(maxlength, temp);}
            else if(x - var01[0] == 0) {maxlength = max(maxlength, temp); continue;}
            else                     {maxlength = max(maxlength, temp); temp = 1; var01[0] = x;}
        }
    }

```

```
    // returning the max-length
    cout << format("final-output = {}\n", maxlength);

    // return
    return(0);

}
```

129. Sum Root to Leaf Numbers

You are given the root of a binary tree containing digits from 0 to 9 only.

Each root-to-leaf path in the tree represents a number. For example, the root-to-leaf path $1 \rightarrow 2 \rightarrow 3$ represents the number 123. Return the total sum of all root-to-leaf numbers. Test cases are generated so that the answer will fit in a 32-bit integer.

Constraints

- The number of nodes in the tree is in the range [1, 1000].
- $0 \leq \text{Node.val} \leq 9$
- The depth of the tree will not exceed 10.

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root           {new TreeNode(4)};
    root->left          = new TreeNode(9);
    root->right         = new TreeNode(0);
    root->left->left    = new TreeNode(5);
    root->left->right = new TreeNode(1);

    // trivial case
```

```cpp
    if (root == nullptr) return 0;

    // setup
    auto runningsum {0};
    auto finalSum  {0};
    std::function<void(const TreeNode*, int)> foo = [&foo,
                                &finalSum](
        const TreeNode*   root,
        int               runningsum){

        // sending it back
        if (root == nullptr) return;

        // adding current value to running sum
        runningsum = runningsum*10 + root->val;

        // going down left or right
        if (root->left || root->right){
            if (root->left)   foo(root->left,   runningsum);
            if (root->right)  foo(root->right,  runningsum);
        }
        else{
            // adding to final sum
            finalSum += runningsum;
        }

        // return re
        return;
    };

    // calling function
    foo(root, runningsum);

    // retunring finalSum
    cout << format("final-output = {}\n", finalSum);
```

```

    // return
    return(0);

}
```

134. Gas Station

There are n gas stations along a circular route, where the amount of gas at the ith station is gas[i]. You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from the ith station to its next (i + 1)th station. You begin the journey with an empty tank at one of the gas stations. Given two integer arrays gas and cost, return the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return -1. If there exists a solution, it is guaranteed to be unique.

Examples

1. **Example 1:**
    - Input: gas = [1,2,3,4,5], cost = [3,4,5,1,2]
    - Output: 3
2. **Example 2:**
    - Input: gas = [2,3,4], cost = [3,4,3]
    - Output: -1

Constraints:

- n == gas.length == cost.length
- $1 \leq n \leq 10^5$
- $0 \leq gas[i], cost[i] \leq 10^4$
- The input is generated such that the answer is unique.

Code

```
int main(){

    // input- configuration
    vector<int> gas   {1,2,3,4,5};
    vector<int> cost  {3,4,5,1,2};

    // setup
    auto acc   {0};                                                  // variable to accumulate values
    vector<int> diffvec;                                             // to store differences
    auto temp  {0};                                                  // to maintain values in cache
    int finaloutput {-1};                                            // the final output sotring var

    // running through it
    for(int i = 0; i<cost.size(); ++i){
        temp   = gas[i] - cost[i];                                   // calculating flux
        acc    += temp;                                              // appending to integral flux
        diffvec.push_back(temp);                                     // storing instantial-flux to vector
    }
    if (acc<0) {finaloutput = -1; return 0;}                         // if total flux is zero, the task cannot be completed

    // going through the diff-vec
    acc = 0;
    for(int i = 0; i<diffvec.size(); ++i){
        acc += diffvec[i];                                           // accumulating flux
        if (acc<0) {acc = 0; finaloutput = i+1;}                     // updating start-point (since we cannot go below zero)
    }


    // printing the acc
    cout << format("acc = {}\n", finaloutput);

    // return
    return(0);
```

```
}
```

135. Candy

There are n children standing in a line. Each child is assigned a rating value given in the integer array ratings. You are giving candies to these children subjected to the following requirements:

1. Each child must have at least one candy.
2. Children with a higher rating get more candies than their neighbors.
3. Return the minimum number of candies you need to have to distribute the candies to the children.

Examples

- **Example 1**
  - Input: ratings = [1,0,2]
  - Output: 5
  - Explanation: You can allocate to the first, second and third child with 2, 1, 2 candies respectively.
- **Example 2**
  - Input: ratings = [1,2,2]
  - Output: 4
  - Explanation: You can allocate to the first, second and third child with 1, 2, 1 candies respectively. The third child gets 1 candy because it satisfies the above two conditions.

Constraints

1. n == ratings.length
2. $1 \leq n \leq 2 * 10^4$
3. $0 \leq ratings[i] \leq 2 * 10^4$

Code

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

    // input- configuration
    vector<int> ratings {1,0,2};

    // setup
    auto candies      {std::vector<int>(ratings.size(),1)};
    auto finaloutput  {static_cast<int>(candies.size())};
    int leftrating, currrating, rightrating;

    // left-pass
    for(int i = 1; i<candies.size(); ++i){

        // fetching the rating
        leftrating = ratings[i-1];
        currrating = ratings[i];

        // fetching references to candy counts
        int& leftcount = candies[i-1];
        int& currcount = candies[i];

        // updating based on left
```

```
        if (currrating > leftrating){
            currcount = leftcount+1;
        }
    }

    // right pass
    for(int i = ratings.size()-2; i>=0; --i){

        // fetching ratings
        currrating = ratings[i];
        rightrating = ratings[i+1];

        // fetching references to candies
        int& currcandies  = candies[i];
        int& rightcandies = candies[i+1];

        // updating based on right
        if (currrating > rightrating){
            currcandies = std::max(currcandies,
                                   rightcandies + 1);
        }
    }

    // summing up candies
    finaloutput = std::accumulate(candies.begin(), candies.end(), 0);
    cout << format("finaloutput = {}\n", finaloutput);

    // return
    return(0);

}
```

141. Linked List Cycle

Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

Return true if there is a cycle in the linked list. Otherwise, return false.

Examples

1. **Example 1:**
    - Input: head = [3,2,0,-4], pos = 1
    - Output: true
    - Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).
2. **Example 2:**
    - Input: head = [1,2], pos = 0
    - Output: true
    - Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.
3. **Example 3:**
    - Input: head = [1], pos = -1
    - Output: false
    - Explanation: There is no cycle in the linked list.

Constraints

- The number of the nodes in the list is in the range [0, 104].
- $-10^5 \leq \text{Node.val} \leq 10^5$
- pos is -1 or a valid index in the linked-list.

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    ListNode* head                  = new ListNode(3);
    head->next                      = new ListNode(2);
    head->next->next                = new ListNode(0);
    head->next->next->next          = new ListNode(-4);
    head->next->next->next->next = head->next;

    // trivial case
    if (head == nullptr) {cout << format("final-output = false\n"); return 0;}

    // setup
    ListNode* dummy   = new ListNode(0);
    dummy->next       = head;
    ListNode* fastboi = dummy;
    ListNode* slowboi = dummy;

    while (fastboi != nullptr && fastboi->next != nullptr) {

```

```
        // updating positions
        fastboi = fastboi->next->next;
        slowboi = slowboi->next;

        // checking if they're the same
        if (slowboi == fastboi) {cout << format("final-output = true\n"); return 0;}
    }

    // since exiting the list implies you left
    cout << format("final-output = false\n"); return 0;

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

151. Reverse Words In A String

Given an input string s, reverse the order of the words. A word is defined as a sequence of non-space characters. The words in s will be separated by at least one space. Return a string of the words in reverse order concatenated by a single space. Note that s may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

Examples

1. **Example 1**
   - Input: s = ”the sky is blue”
   - Output: ”blue is sky the”
2. **Example 2**
   - Input: s = ” hello world ”
   - Output: ”world hello”
   - Explanation: Your reversed string should not contain leading or trailing spaces.
3. **Example 3**
   - Input: s = ”a good example”
   - Output: ”example good a”
   - Explanation: You need to reduce multiple spaces between two words to a single space in the reversed string.

Constraints

1. $1 \leq \text{s.length} \leq 10^4$
2. s contains English letters (upper-case and lower-case), digits, and spaces ' '.
3. There is at least one word in s.

Code

```cpp
int main(){

    // input- configuration
    string s   {"a good  example"};

    // setup
    vector<string> listofwords;

    // creating a list of words
    int p1 {0};
    string acc;
    while(p1 < s.size()){

        // checking if the current character is a non-space
        if (s[p1] != ' ') {acc += s[p1];}
        else{
            // if acc is non-empty, flush
            if (acc.size() != 0)  {listofwords.push_back(acc); acc = "";}
            else                  {;}
        }

        // moving the index-pointer forward
        p1++;
```

```
    }

    // check if acc is unflushed
    if (acc.size() != 0)  {listofwords.push_back(acc); acc = "";}

    // building the finaloutput
    string finaloutput;
    for(int i = listofwords.size()-1; i>=0; --i){
        finaloutput += listofwords[i];
        if (i!=0) [[unlikely]] {finaloutput += " ";}
    }

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


    // return
    return(0);

}
```

167. Two Sum II - Input Array Is Sorted

Given a 1-indexed array of integers numbers that is already sorted in non-decreasing order, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where $1 \leq$ index1 $<$ index2 $\leq$ numbers.length.

Return the indices of the two numbers, index1 and index2, added by one as an integer array [index1, index2] of length 2.

The tests are generated such that there is exactly one solution. You may not use the same element twice.

Your solution must use only constant extra space.

Examples

1. **Example 1:**
    - Input: numbers = [2,7,11,15], target = 9
    - Output: [1,2]
    - Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].
2. **Example 2:**
    - Input: numbers = [2,3,4], target = 6
    - Output: [1,3]
    - Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3].
3. **Example 3:**
    - Input: numbers = [-1,0], target = -1

- Output: [1,2]
- Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].

Constraints

- $2 \leq$ numbers.length $\leq 3 * 10^4$
- -1000 $\leq$ numbers[i] $\leq$ 1000
- numbers is sorted in non-decreasing order.
- -1000 $\leq$ target $\leq$ 1000
- The tests are generated such that there is exactly one solution.

Code

```
int main(){

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

    // setup
    std::vector<int> finalOutput;
    auto left     {0};
    auto right    {static_cast<int>(numbers.size()-1)};
    auto currsum  {0};

    // usual left-right loop
    while (left < right){
```

```

        // checking sum of two values
        currsum = numbers[left] + numbers[right];

        // comparing against target
        if (currsum > target)    {--right;}
        else if (currsum < target) {++left;}
        else {
            finalOutput.push_back( left+1);
            finalOutput.push_back( right+1);
            break;
        }
    }

    // printign the final output
    cout << format("finaloutput = {}\n", finalOutput);

    // return
    return(0);

}
```

169 Majority Element

Given an array nums of size n, return the majority element. The majority element is the element that appears more than $\lfloor n/2 \rfloor$ times. You may assume that the majority element always exists in the array.

Examples

- **Example 1**
  - `Input: nums = [3,2,3]`
  - `Output: 3`
- **Example 2**
  - `Input: nums = [2,2,1,1,1,2,2]`
  - `Output: 2`

Constraints:

- n == nums.length
- $1 \leq \text{n} \leq 5 * 10^4$
- $-10^9 \leq \text{nums[i]} \leq 10^9$

Code

```
int main(){

    // input- configuration
    vector<int> nums  {2,2,1,1,1,2,2};

    // setup
    unordered_map<int, int> histogram;
    int max_element   {std::numeric_limits<int>::min()};
    int max_count     {std::numeric_limits<int>::min()};
    int updated_count {0};

    // going through the elements
    for(int i = 0; i<nums.size(); ++i){

        // adding to histogram
        if (histogram.find(nums[i]) == histogram.end()) {histogram[nums[i]] = 1; updated_count = 0;}
        else                                          {++histogram[nums[i]]; updated_count = histogram[nums[i]];}

        // keeping track of max-element
        if (updated_count > max_count) {max_element = nums[i]; max_count = updated_count;}

    }

    // printing the final output
    cout << format("nums = "); fpv(nums);
    cout << format("max-count = {}\n", max_count);

    // return
    return(0);

}
```

189 Rotate Array

Given an integer array nums, rotate the array to the right by k steps, where k is non-negative.

Examples

- **Example 1**
  - Input: nums = [1,2,3,4,5,6,7], k = 3
  - Output: [5,6,7,1,2,3,4]
- **Example 1**
  - Input: nums = [-1,-100,3,99], k = 2
  - Output: [3,99,-1,-100]

Constraints

- $1 \leq \text{nums.length} \leq 10^5$
- $-2^31 \leq \text{nums[i]} \leq 2^31 - 1$
- $0 \leq k \leq 10^5$

Code

```
int main(){

    // input- configuration
    vector<int> nums  {-1,-100,3,99};
    int k  {2};

    // setup
    StopWatch timer;                                     // setting up the timer
    k = k %static_cast<int>(nums.size());                // to ensure that the value is within range

    int source        {0};
    int temp_source   {nums[source]};
    int temp          {0};
    int destination   {0};

    vector<bool> sourcelist(nums.size(), false);

    // going through nums
    for(int i = 0; i < nums.size(); ++i){

        // check if curent-source has been taken care of
        if (sourcelist[source] == true){
            source     = (source+1) % nums.size();
            temp_source = nums[source];
        }

        source            = source % nums.size();       // code to ensure range
        destination       = (source + k)%nums.size();   // calculating the index we'll be writing to
        sourcelist[source] = true;                       // updating source-list

        temp              = nums[destination];          // safe-keeping the destination value
        nums[destination] = temp_source;                 // storing new value at destination-index

```

```
        source              = destination;                  // updating source-index
        temp_source         = temp;                         // updating source-value
    }

    // printing the output
    cout << format("nums = "); fpv(nums);                   // printing the updated array, "nums"
    timer.stop();                                           // printing the time taken

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

200. Number of Islands

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Examples

1. **Example 1:**
   - Input:

1	1	1	1	0
1	1	0	1	0
1	1	0	0	0
0	0	0	0	0

   - Output: 1
2. **Example 2:**
   - Input:

1	1	0	0	0
1	1	0	0	0
0	0	1	0	0
0	0	0	1	1

   - Output: 3

Constraints

- m == grid.length

- n == grid[i].length
- $1 \leq$ m, n ≤ 300
- grid[i][j] is '0' or '1'.

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    vector<vector<string>> grid {
        vector<string>({"1","1","1","1","0"}),
        vector<string>({"1","1","0","1","0"}),
        vector<string>({"1","1","0","0","0"}),
        vector<string>({"0","0","0","0","0"})
    };

    // setup
    std::vector< std::vector<bool> > didWeVisitThisPlace;
    for(int i = 0; i<grid.size(); ++i)
        didWeVisitThisPlace.push_back(std::vector<bool>(grid[0].size(), false));

    // lambda to check validity
    auto fCheckValidityOfCoordinate = [&grid](
        std::vector<int> coords){

        // spreading it out
        auto row  {coords[0]};
        auto col  {coords[1]};
```

```
        // checking validity
        if (row >= 0          &&
            row < grid.size() &&
            col >= 0          &&
            col <grid[0].size())
            if (grid[row][col] == "1") {return true;}

        // in case the above condition is not met
        return false;

    };

    // traversal function
    std::function<void(int, int)> fMarkThemAll = [&fMarkThemAll,
                                                  &grid,
                                                  &didWeVisitThisPlace,
                                                  fCheckValidityOfCoordinate](
        int row_index,
        int col_index){

        // setting up coordinates
        std::vector<int> rightCoordinate({row_index, col_index+1});
        std::vector<int> downCoordinate{row_index+1, col_index};

        // marking the current coordinate as visited
        didWeVisitThisPlace[row_index][col_index] = true;

        // calling the function to the right
        if (fCheckValidityOfCoordinate(rightCoordinate) == true &&\
            didWeVisitThisPlace[rightCoordinate[0]][rightCoordinate[1]] == false)
                fMarkThemAll(rightCoordinate[0], rightCoordinate[1]);

        // calling the function for the ones below
        if (fCheckValidityOfCoordinate(downCoordinate) == true && \
```

```
            didWeVisitThisPlace[downCoordinate[0]][downCoordinate[1]] == false)
                fMarkThemAll(downCoordinate[0], downCoordinate[1]);
    };


    // going through the elements
    int count = 0;
    for(int row_index = 0; row_index < grid.size(); ++row_index){
        for(int col_index = 0; col_index < grid[0].size(); ++col_index){
            // starting an exploratory course if this point has not been visited
            if (didWeVisitThisPlace[row_index][col_index] == false && \
                grid[row_index][col_index]              == "1"){
                fMarkThemAll(row_index, col_index);
                ++count;
            }
        }
    }

    // return count
    cout << format("final-output = {}\n", count);

    // return
    return(0);

}
```

202. Happy Number

Write an algorithm to determine if a number n is happy.

A happy number is a number defined by the following process:

1. Starting with any positive integer, replace the number by the sum of the squares of its digits.
2. Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.
3. Those numbers for which this process ends in 1 are happy.

Return true if n is a happy number, and false if not.

Examples

1. **Example 1:**
    - Input: n = 19
    - Output: true
    - Explanation:
        - 12 + 92 = 82
        - 82 + 22 = 68
        - 62 + 82 = 100
        - 12 + 02 + 02 = 1
2. **Example 2:**
    - Input: n = 2
    - Output: false

Constraints

- $1 \leq \mathrm{n} \leq 2^3 1 - 1$

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto n {19};

    // trivial case
    if (n == 4 || n == 0) {cout << format("final-output = false\n"); return 0;}

    // setting up lambda
    auto fSumDigitSquares = [](int n) -> int
    {
        auto sum   {0};
        auto digit {-1};
        while(n!=0){
            digit  = n%10;
            sum    += digit*digit;
            n      = n/10;
        }
        return sum;
    };

    // calling the function
    while (n != 1){
        // calculating digits sums
```

```
        n = fSumDigitSquares(n); cout << format("n = {}\n", n);
        if (n == 4 || n == 0)    {cout << format("final-output = false\n"); return 0;}
    }

    // printing and returning
    cout << format("final-output = true \n"); return 0;

}
```

205. Isomorphic Strings

Given two strings s and t, determine if they are isomorphic.

Two strings s and t are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

Examples

1. **Example 1:**
    - Input: s = ”egg”, t = ”add”
    - Output: true
    - Explanation: The strings s and t can be made identical by:
        - Mapping ’e’ to ’a’.
        - Mapping ’g’ to ’d’.
2. **Example 2:**
    - Input: s = ”foo”, t = ”bar”
    - Output: false
    - Explanation: The strings s and t can not be made identical as ’o’ needs to be mapped to both ’a’ and ’r’.
3. **Example 3:**
    - Input: s = ”paper”, t = ”title”
    - Output: true

Constraints:

- $1 \leq \text{s.length} \leq 5 * 10^4$
- t.length == s.length
- s and t consist of any valid ascii character.

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    string s = {"egg"};
    string t = {"add"};

    // making mapping
    unordered_map<char, char> mapping;
    std::unordered_set<char> destinations;
    string                    r;

    // going through string
    for(int i = 0; i<s.size(); ++i){

        // checking if two strings are different
        if (s[i] != t[i]){
            // checking if we already have a mapping for this
            if (mapping.find(s[i]) == mapping.end()){

                // before we addd, we need to check if t[i] is already used as a destination
```

```
            if (destinations.find(t[i]) != destinations.end()) {
                cout << format("final-output = true \n");
                return 0;
            }

            // adding new mapping
            mapping[s[i]] = t[i];
            destinations.insert(t[i]);
        }
        r.push_back(mapping[s[i]]);
    }
    else{
        // checking if we already have a mapping for this
        if (mapping.find(s[i]) == mapping.end()){

            // before we addd, we need to check if t[i] is already used as a destination
            if (destinations.find(t[i]) != destinations.end()) {
                cout << format("final-output = true \n");
                return 0;
            }

            // adding new mapping
            mapping[s[i]] = s[i];
            destinations.insert(s[i]);
        }
        r.push_back(mapping[s[i]]);
    }

    // if what we have so far is different, we're sending it back
    string s_subset(t.begin(), t.begin() + r.size());
    if(s_subset != r) {cout << format("final-output = false \n"); return 0;}

}

// return true in the end
```

```
    cout << format("final-output = true \n");

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

209. Minimum Size Subarray Sum

Given an array of positive integers nums and a positive integer target, return the minimal length of a subarray whose sum is greater than or equal to target. If there is no such subarray, return 0 instead.

Examples

1. **Example 1:**
   - Input: target = 7, nums = [2,3,1,2,4,3]
   - Output: 2
   - Explanation: The subarray [4,3] has the minimal length under the problem constraint.
2. **Example 2:**
   - Input: target = 4, nums = [1,4,4]
   - Output: 1
3. **Example 3:**
   - Input: target = 11, nums = [1,1,1,1,1,1,1,1]
   - Output: 0

Constraints

- $1 \leq \text{target} \leq 10^9$

- $1 \leq \text{nums.length} \leq 10^5$
- $1 \leq \text{nums[i]} \leq 10^4$

Code

```
int main(){

    // input- configuration
    vector<int> nums  {1,2,3,4,5};
    auto target    {15};

    // setup
    auto finaloutput  {std::numeric_limits<int>::max()};
    auto pleft        {0};
    auto sum          {nums[pleft]};
    auto i            {1};

    // in case the first element itself is greater
    if (sum > target) {finaloutput = 1;}

    // lambda to update finaloutput
    auto updatefinaloutput = [&sum,
                              &target,
                              &finaloutput,
                              &i,
                              &pleft,
                              &nums]() -> void{

        auto numelements = i - pleft + 1;
        finaloutput = numelements < finaloutput ? numelements : finaloutput;
        sum -= nums[pleft++];
    };
```

```

    // going through the array
    for(; i<nums.size(); ++i){

        // adding to sum
        sum += nums[i];

        // updating
        while (sum>=target) {updatefinaloutput();}
    }

    // updating
    if(finaloutput == std::numeric_limits<int>::max()) {finaloutput = 0;}

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

    // return
    return(0);

}
```

216. Combination Sum III

Find all valid combinations of k numbers that sum up to n such that the following conditions are true:

- Only numbers 1 through 9 are used.
- Each number is used at most once.

Return a list of all possible valid combinations. The list must not contain the same combination twice, and the combinations may be returned in any order.

Examples

1. **Example 1:**
    - Input: k = 3, n = 7
    - Output: [[1,2,4]]
    - Explanation:
        - 1 + 2 + 4 = 7
        - There are no other valid combinations.
2. **Example 2:**
    - Input: k = 3, n = 9
    - Output: [[1,2,6],[1,3,5],[2,3,4]]

- Explanation:
  - 1 + 2 + 6 = 9
  - 1 + 3 + 5 = 9
  - 2 + 3 + 4 = 9
  - There are no other valid combinations.

3. **Example 3:**

   - Input: k = 4, n = 1
   - Output: []
   - Explanation: There are no valid combinations.
     - Using 4 different numbers in the range [1,9], the smallest sum we can get is 1+2+3+4 = 10 and since 10 ¿ 1, there are no valid combination.

Constraints

- $2 \leq k \leq 9$
- $1 \leq n \leq 60$

Code

```cpp
void fTraverse(std::vector<int> numberPath, \
               int target, \
               int k, \
               std::vector< std::vector<int> >& finalOutput){

    // checking if length is too much
    if (numberPath.size() > k) return;
```

```
    // calculating sum so far
    int sumsofar = std::accumulate(numberPath.begin(), numberPath.end(),0);

    // number to fill
    int sumtofill = target - sumsofar;

    // if this is zero, we can add it to list and just sendn it back
    if (sumtofill == 0){
        if (numberPath.size() == k){
            if (std::find(finalOutput.begin(),
                          finalOutput.end(),
                          numberPath) == finalOutput.end())
                finalOutput.push_back(numberPath);                          // add if it doesn't already exist
        }

        return;
    }

    // valid candidates
    std::vector<int> candidates;
    int biggest_number = 1;
    if (numberPath.size()!=0){
        auto iter = std::max_element(numberPath.begin(), numberPath.end());
        biggest_number = *iter;
    }

    sumtofill = std::min(sumtofill, 9);
    for(int i = biggest_number; i<=sumtofill; ++i){
        // add if no on th epath so far
        if (std::find(numberPath.begin(),
                      numberPath.end(),
                      i) == numberPath.end()) {candidates.push_back(i);}
    }

```

```
    // if there are no candidates, we're going back
    if (candidates.size() == 0) return;

    // trying each candidate
    std::vector<int> numberPath_local;
    for(auto x: candidates){
        // sending down each candidate route
        numberPath_local = numberPath;
        numberPath_local.push_back(x);
        fTraverse(numberPath_local, target, k, finalOutput);
    }

    // returning
    return;
}

int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto k {3};
    auto n {7};

    // setup
    std::vector< std::vector<int> > finalOutput;
    std::vector<int> numberPath;

    // recursion
    fTraverse(numberPath, n, k, finalOutput);

    // printing
    cout << format("final-output = {}\n", finalOutput);

```

```
    // return
    return(0);

}
```

219. Contains Duplicate II

Given an integer array nums and an integer k, return true if there are two distinct indices i and j in the array such that nums[i] == nums[j] and abs(i - j) $\leq$ k.

Examples

1. **Example 1:**
   - Input: nums = [1,2,3,1], k = 3
   - Output: true
2. **Example 2:**
   - Input: nums = [1,0,1,1], k = 1
   - Output: true
3. **Example 3:**
   - Input: nums = [1,2,3,1,2,3], k = 2
   - Output: false

Constraints

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums[i]} \leq 10^9$
- $0 \leq k \leq 10^5$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto nums = std::vector<int>({1,2,3,1,2,3});
    auto k     {2};

    // setup
    std::unordered_map<int, int> valueToIndex;

    // going through the array
    for(int i = 0; i<nums.size(); ++i){

        // adding current element to the hamp
        if (valueToIndex.find(nums[i]) == valueToIndex.end()){
            // if it doesn't exist, we add to it
            valueToIndex[nums[i]] = i;
        }
        else{
            // if it already exists, calculating distance from the first index
            if (i - valueToIndex[nums[i]] <= k) {cout << format("final-output = true\n");}
            else                              {valueToIndex[nums[i]] = i;}
        }
    }

    // returning false in the final case
    cout << format("final-output = false\n");

    // return
    return(0);

```

```
}
```

226. Invert Binary Tree

Given the root of a binary tree, invert the tree, and return its root.

Examples

1. **Example 1:**
    - Input: root = [4,2,7,1,3,6,9]
    - Output: [4,7,2,9,6,3,1]
2. **Example 2:**
    - Input: root = [2,1,3]
    - Output: [2,3,1]
3. **Example 3:**
    - Input: root = []
    - Output: []

Constraints

- The number of nodes in the tree is in the range [0, 100].
- -100 $\leq$ Node.val $\leq$ 100

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root               {new TreeNode(4)};
    root->left               = new TreeNode(2);
    root->right              = new TreeNode(7);
    root->left->left         = new TreeNode(1);
    root->left->right        = new TreeNode(3);
    root->right->left        = new TreeNode(6);
    root->right->right       = new TreeNode(9);

    // setup
    std::function<void(TreeNode*)> fTwist = [&fTwist](TreeNode* root){
        // base-case
        if (root == nullptr) return;

        // switching left and right
        fTwist(root->left);
        fTwist(root->right);

        // twisting current branches
        auto temp  = root->left;
        root->left = root->right;
        root->right = temp;

        // returning
        return;
    };

    // going flipping through eveyrthing
```

```
    fTwist(root);

    // return
    return(0);

}
```

228. Summary Ranges

You are given a sorted unique integer array nums.

A range [a,b] is the set of all integers from a to b (inclusive).

Return the smallest sorted list of ranges that cover all the numbers in the array exactly. That is, each element of nums is covered by exactly one of the ranges, and there is no integer x such that x is in one of the ranges but not in nums.

Each range [a,b] in the list should be output as:

"a→b" if a != b

"a" if a == b

Examples

1. **Example 1:**

    - Input: nums = [0,1,2,4,5,7]
    - Output: ["0→2","4→5","7"]
    - Explanation: The ranges are:
        - [0,2] $\implies$ "0→2"
        - [4,5] $\implies$ "4→5"
        - [7,7] $\implies$ "7"

2. **Example 2:**

    - Input: nums = [0,2,3,4,6,8,9]
    - Output: ["0","2→4","6","8→9"]

- Explanation: The ranges are:
  - [0,0] $\implies$ ”0”
  - [2,4] $\implies$ ”2$\rightarrow$4”
  - [6,6] $\implies$ ”6”
  - [8,9] $\implies$ ”8$\rightarrow$9”

Constraints

- $0 \leq \text{nums.length} \leq 20$
- $-2^31 \leq \text{nums[i]} \leq 2^31 - 1$
- All the values of nums are unique.
- nums is sorted in ascending order.

Code

```
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto nums  = vector<int>({0,1,2,4,5,7});

    // trivial-case
    if (nums.size() == 0) {cout << format("final-output = {}\n", vector<string>()); return 0;}

    // stup
```

```
    auto arrow {"->"};
    auto p1    {0};
    auto p2    {0};
    vector<string> finalOutput;

    // while loop
    while (p2<nums.size()){

        // checking prev element
        if (nums[p2] <= 1 + nums[p2-1]) {++p2;}
        else{
            if (p1!=p2-1) {finalOutput.push_back(std::to_string(nums[p1]) + arrow + std::to_string(nums[p2-1]));}
            else          {finalOutput.push_back(std::to_string(nums[p2-1]));}
            p1 = p2++;
        }
    }

    // checking if p1 and p2 areright ne
    if (p1!=p2-1) {finalOutput.push_back(std::to_string(nums[p1]) + arrow + std::to_string(nums[p2-1]));}
    else          {finalOutput.push_back(std::to_string(nums[p2-1]));}

    // returning restul
    cout << format("final-output = {}\n", finalOutput);

    // return
    return(0);

}
```

230. Kth Smallest Element in a BST

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

Constraints

- The number of nodes in the tree is n.
- $1 \leq k \leq n \leq$
- $0 \leq \text{Node.val} \leq 10^4$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root           {new TreeNode(3)};
    root->left          = new TreeNode(1);
    root->right         = new TreeNode(4);
    root->left->right = new TreeNode(2);
    auto k              {1};

    // setup
    int finalOutput;
    auto stopsearch {false};
```

```
    auto leftindex {0};
    std::function<void(const TreeNode*, const int)> foo = [&foo, &leftindex, &finalOutput, &stopsearch](
        const TreeNode* root,
        const int k){
        // sending it back
        if (root == nullptr)  {return;}

        // checking left
        if (root->left)       {foo(root->left, k);}

        // appending count
        ++leftindex;
        if (leftindex == k)   {finalOutput = root->val; stopsearch = true;}

        // going right
        if (root->right)      {foo(root->right, k);}

        // goign back
        return;
    };

    // running
    foo(root, k);

    // printing otuput
    cout << format("final-output = {}\n", finalOutput);

    // return
    return(0);

}
```

238. Product of Array Except Self

Given an integer array nums, return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i]. The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer. You must write an algorithm that runs in O(n) time and without using the division operation.

Examples

1. **Example 1**
   - Input: nums = [1,2,3,4]
   - Output: [24,12,8,6]
2. **Example 2**
   - Input: nums = [-1,1,0,-3,3]
   - Output: [0,0,9,0,0]

Constraints

1. $2 \leq$ nums.length $\leq 10^5$
2. -30 $\leq$ nums[i] $\leq$ 30
3. The input is generated such that answer[i] is guaranteed to fit in a 32-bit integer

Code

```
int main(){

    // input- configuration
    vector<int> nums  {1,2,3,4};

    // setup
    vector<int> nums_left(nums.size(), 1);
    vector<int> nums_right(nums.size(), 1);
    int acc_left          {1};
    int acc_right         {1};

    // runs
    for(int i = 0; i<nums.size(); ++i){

        // source-indces
        int source_left {i-1};
        int source_right {static_cast<int>(nums.size())-i};

        // printing values
        acc_left   *= source_left == -1        ? 1 : nums[source_left];
        acc_right  *= source_right == nums.size() ? 1 : nums[source_right];

        // writing to the two values
        nums_left[i]                 = acc_left;
        nums_right[nums.size()-i-1]  = acc_right;
    }

    // building the accumlated value
    vector<int> finaloutput(nums.size(),1);
    for(int i = 0; i< finaloutput.size(); ++i){
        finaloutput[i] = nums_left[i] * nums_right[i];
    }

```

```
    // printing
    cout << format("finaloutput = "); fPrintVector(finaloutput);

    // return
    return(0);

}
```

242. Valid Anagram

Given two strings s and t, return true if t is an anagram of s, and false otherwise.

Examples

1. **Example 1:**
   - Input: s = ”anagram”, t = ”nagaram”
   - Output: true
2. **Example 2:**
   - Input: s = ”rat”, t = ”car”
   - Output: false

Constraints

- $1 \leq$ s.length, t.length $\leq 5 * 10^4$
- s and t consist of lowercase English letters.

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    string s   {"anagram"};
    string t   {"nagaram"};

    // setup
    std::vector<int> s_count(26,0);
    std::vector<int> t_count(26,0);

    // building count-vector for both
    for(auto x: s) ++s_count[static_cast<int>(x)-97];
    for(auto x: t) ++t_count[static_cast<int>(x)-97];

    // comparing the two
    for (int i = 0; i<s_count.size(); ++i) {

        // element-wise checking
        if (s_count[i] != t_count[i]) {cout << format("final-output = false \n");}

    }

    // returning
    cout << format("final-output = true \n");

    // return
    return(0);

}
```

274. H-Index

Given an array of integers citations where citations[i] is the number of citations a researcher received for their ith paper, return the researcher's h-index. According to the definition of h-index on Wikipedia: The h-index is defined as the maximum value of h such that the given researcher has published at least h papers that have each been cited at least h times.

Examples

1. **Example 1**
   - Input: citations = [3,0,6,1,5]
   - Output: 3
   - Explanation: [3,0,6,1,5] means the researcher has 5 papers in total and each of them had received 3, 0, 6, 1, 5 citations respectively. Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, their h-index is 3.
2. **Example 2**
   - Input: citations = [1,3,1]
   - Output: 1

Constraints

1. n == citations.length
2. $1 \leq n \leq 5000$
3. $0 \leq citations[i] \leq 1000$

Code

```
int main(){

    // input- configuration
    vector<int> citations {3,0,6,1,5};

    // sorting the citations first
    std::sort(citations.begin(), citations.end(),
        [](const int& a, const int& b) {return a>b;});

    // running accumulations
    auto hvalue    {0};
    for(int i = 0; i<citations.size(); ++i){
        if (citations[i] >= (i+1))   {hvalue = i+1;}
    }

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

    // return
    return(0);

}
```

283. Move Zeros

Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements. Note that you must do this in-place without making a copy of the array.

Examples

1. **Example 1:**
   - Input: nums = [0,1,0,3,12]
   - Output: [1,3,12,0,0]
2. **Example 2:**
   - Input: nums = [0]
   - Output: [0]

Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $-2^{31} \leq \text{nums[i]} \leq 2^{31} - 1$

Code

```cpp
int main(){

    // input- configuration
    vector<int> nums  {0,1,0,3,12};

    // setup
    int explorer   {0};
    int anchor     {0};

    // going through the nums
    while(explorer < nums.size()){

        // moving explorer until we arrive at a non-zero value
        while(explorer < nums.size() && nums[explorer] == 0) {explorer++;}

        // copying value
        if (explorer<nums.size() && anchor <nums.size())
            nums[anchor++]  = nums[explorer++];
    }

    // zeroing out the rest
    while(anchor < nums.size()) {nums[anchor++] = 0;}

    // printing the finaloutput
    cout << format("finaloutput = "); fPrintVector(nums);

    // return
    return(0);

}
```

345. Reverse Vowels Of A String

Given a string s, reverse only all the vowels in the string and return it. The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in both lower and upper cases, more than once.

Examples

1. **Example 1:**
   - Input: s = "IceCreAm"
   - Output: "AceCreIm"
   - Explanation: The vowels in s are ['I', 'e', 'e', 'A']. On reversing the vowels, s becomes "AceCreIm".
2. **Example 2:**
   - Input: s = "leetcode"
   - Output: "leotcede"

Constraints

- $1 \leq \text{s.length} \leq 3 * 10^5$
- s consist of printable ASCII characters.

Code

```cpp
int main(){

    // input- configuration
    string s   {"leetcode"};

    // going through the string
    string     vowels        {"aeiouAEIOU"};
    vector<int> vowel_indices;
    string     reversed_vowels;
    string     finaloutput = s;

    // going through the string
    for(int i = 0; i<s.size(); ++i){
        if (vowels.find(s[i]) != string::npos){
            reversed_vowels+=s[i];
            vowel_indices.push_back(i);
        }
    }

    // refilling the indices
    for(int i = 0; i<vowel_indices.size(); ++i){
        finaloutput[vowel_indices[i]] = reversed_vowels[reversed_vowels.size()-1-i];
    }

    // printing the final output
    cout << format("finaloutput = {}\n", finaloutput);

    // 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).

Examples

1. **Example 1:**
    - Input: s = ”abc”, t = ”ahbgdc”
    - Output: true
2. **Example 2:**
    - Input: s = ”axc”, t = ”ahbgdc”
    - Output: false

Constraints

- $0 \leq \text{s.length} \leq 100$
- $0 \leq \text{t.length} \leq 10^4$
- s and t consist only of lowercase English letters.

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);

}
```

394. Decode String

Given an encoded string, return its decoded string.

The encoding rule is: `k[encoded_string]`, where the `encoded_string` inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer.

You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there will not be input like 3a or 2[4].

The test cases are generated so that the length of the output will never exceed 10^5.

Examples

1. **Example 1:**
   - Input: s = ”3[a]2[bc]”
   - Output: ”aaabcbc”
2. **Example 2:**
   - Input: s = ”3[a2[c]]”
   - Output: ”accaccacc”
3. **Example 3:**
   - Input: s = ”2[abc]3[cd]ef”
   - Output: ”abcabccdcdcdef”

Constraints:

- $1 \leq$ s.length ≤ 30
- s consists of lowercase English letters, digits, and square brackets '[]'.
- s is guaranteed to be a valid input.
- All the integers in s are in the range [1, 300].

Code

```cpp
int main(){

    // input- configuration
    string s   {"100[leetcode]"};

    // running
    std::stack<char>  mystack;                                          // stack
    string            temp;                                             // temporary string used for decoding
    int               repcount   {1};                                   // used for decoding

    // going through the inputs
    for(int i = 0; i<s.size(); ++i){

        if(s[i] != ']')   {mystack.push(s[i]);}                         // pushing characters to stack until we arrive at "]"
        else{
            temp = "";                                                  // initializing temporary string
            while(mystack.top() != '[') {                               // expanding mini-string until we arrive at "["
                temp = mystack.top() + temp;
                mystack.pop();
```

```cpp
        }

        mystack.pop();                                              // removing "["

        // calculating the repcount
        string numberasstring = "";
        while(mystack.size() != 0 &&
              mystack.top() - '0' >= 0 && '9' - mystack.top() >= 0)
        {
            numberasstring = mystack.top() + numberasstring;
            mystack.pop();
        }
        repcount = std::stoi(numberasstring);

        // mini-decoding
        int multitempsize = repcount * temp.size();                 // calculating size after multiplication
        for(int j = 0; j<multitempsize; ++j) {                      // filling up the stack with decoded content
            mystack.push(temp[j%temp.size()]);
        }
    }
}

// creating the final output
string finaloutput;
while(mystack.size()){
    finaloutput = mystack.top() + finaloutput;
    mystack.pop();
}

// printing the final output
cout << format("finaloutput = {}\n", finaloutput);

// return
return(0);
```

```
}
```

433. Minimum Genetic Mutation

A gene string can be represented by an 8-character long string, with choices from 'A', 'C', 'G', and 'T'.

Suppose we need to investigate a mutation from a gene string startGene to a gene string endGene where one mutation is defined as one single character changed in the gene string.

For example, "AACCGGTT" $\rightarrow$ "AACCGGTA" is one mutation.

There is also a gene bank bank that records all the valid gene mutations. A gene must be in bank to make it a valid gene string.

Given the two gene strings startGene and endGene and the gene bank bank, return the minimum number of mutations needed to mutate from startGene to endGene. If there is no such a mutation, return -1.

Note that the starting point is assumed to be valid, so it might not be included in the bank.

Examples

1. **Example 1:**
    - Input: startGene = "AACCGGTT", endGene = "AACCGGTA", bank = ["AACCGGTA"]
    - Output: 1
2. **Example 2:**
    - Input: startGene = "AACCGGTT", endGene = "AAACGGTA", bank = ["AACCGGTA","AACCGCTA","AAACGGTA"]
    - Output: 2

Constraints

- $0 \leq \text{bank.length} \leq 10$
- startGene.length == endGene.length == bank[i].length == 8
- startGene, endGene, and bank[i] consist of only the characters ['A', 'C', 'G', 'T'].

Code

```
void foo(unordered_map<string, vector<string>>& stringtoneighbours,
        vector<string>                          pathsofar,
        bool&                                   foundpath,
        string                                  endGene,
        int&                                    finaloutput){

    // checking if the top of the stack has valid neighbours
    string top = pathsofar[pathsofar.size()-1];

    // checking if the current one is thef inal output
    if (top == endGene)
        finaloutput = finaloutput < pathsofar.size() ? finaloutput : pathsofar.size();

    // checking its possible next-states
    auto nextnodes = stringtoneighbours[top];

    // going depth-first
    for(auto x: nextnodes){

        // not considering if it is already in the path
        if(std::find(pathsofar.begin(),
                    pathsofar.end(),
                    x) != pathsofar.end()) continue;
```

```cpp

        // updating path sof far
        auto pathsofar_temp = pathsofar;
        pathsofar_temp.push_back(x);

        // calling function do it
        foo(stringtoneighbours, pathsofar_temp, foundpath, endGene, finaloutput);
    }
}

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

    // starting timer
    Timer timer;

    // input- configuration
    string startGene      = "AACCGGTT";
    string endGene        = "AACCGGTA";
    vector<string> bank   = {"AACCGGTA"};

    // setup
    unordered_map<string, vector<string>> stringtoneighbours;

    // checking if endgene is in the bank
    if (std::find(bank.begin(),
                  bank.end(),
                  endGene) == bank.end()) {cout << format("finalOutput = -1\n");}

    // going through the bank and building neighbours
    bank.push_back(startGene);

    for(auto x: bank){

        // finding valid transactions with the other strings in the bank
```

```
            for(auto y: bank){

                // checking number of string differences between the two
                auto count = 0;
                for(int i =0; i<8; ++i) {if (x[i] != y[i]) ++count;}

                // checking count and adding to valid transactions
                if (count == 1)       {stringtoneighbours[x].push_back(y);}
            }

            // checking if this particular gene can jump to final gene
            // checking number of string differences between the two
            auto count = 0;
            for(int i =0; i<8; ++i) {if (x[i] != endGene[i]) ++count;}

            // checking count and adding to valid transactions
            if (count == 1)       {stringtoneighbours[x].push_back(endGene);}
        }

        // recursion
        vector<string> pathsofar {startGene};
        bool           foundpath  {false};
        int            finalOutput {-1};

        // calling the function
        foo(stringtoneighbours, pathsofar, foundpath, endGene, finalOutput);

        // returning the finaloutput
        if (finalOutput > -1) --finalOutput;

        // printing final output
        cout << format("finalOutput = {}\n", finalOutput);

        // return
        return(0);
```

```
}
```

443. String Compression

Given an array of characters chars, compress it using the following algorithm:

Begin with an empty string s. For each group of consecutive repeating characters in chars:

1. If the group's length is 1, append the character to s.
2. Otherwise, append the character followed by the group's length.

The compressed string s should not be returned separately, but instead, be stored in the input character array chars. Note that group lengths that are 10 or longer will be split into multiple characters in chars. After you are done modifying the input array, return the new length of the array. You must write an algorithm that uses only constant extra space.

Examples

1. **Example 1:**
   - Input: chars = [”a”,”a”,”b”,”b”,”c”,”c”,”c”]
   - Output: Return 6, and the first 6 characters of the input array should be: [”a”,”2”,”b”,”2”,”c”,”3”]
   - Explanation: The groups are ”aa”, ”bb”, and ”ccc”. This compresses to ”a2b2c3”.
2. **Example 2:**
   - Input: chars = [”a”]
   - Output: Return 1, and the first character of the input array should be: [”a”]
   - Explanation: The only group is ”a”, which remains uncompressed since it's a single character.

3. **Example 3:**
   - Input: chars = ["a","b","b","b","b","b","b","b","b","b","b","b","b"]
   - Output: Return 4, and the first 4 characters of the input array should be: ["a","b","1","2"].
   - Explanation: The groups are "a" and "bbbbbbbbbbbb". This compresses to "ab12".

Constraints

- $1 \leq \text{chars.length} \leq 2000$
- chars[i] is a lowercase English letter, uppercase English letter, digit, or symbol.

Code

```cpp
int main(){

    // input- configuration
    vector<char> chars {'a','b','b','b','b','b','b','b','b','b','b','b','b'};

    // going through the character
    int p1            {0};
    char runningchar  {};
    char curr         {};
    int count         {0};
    string finaloutput;

    // going through the inputs
    while(p1<chars.size()){

        // getting curren tchar
```

```
        curr = chars[p1];

        // increasing count
        if (count == 0)              {runningchar = chars[p1]; ++count;}
        else if(curr == runningchar) {++count;}
        else if(curr != runningchar) {
            finaloutput += runningchar; // writing character to current pointer
            if (count != 1)
                finaloutput += std::to_string(count); // increasing write-pointer
            runningchar = curr;
            count = 1;
        }

        // increasing pointer
        ++p1;
    }

    // flushing out
    if (count != 0){
        finaloutput += runningchar; // writing character to current pointer
        if (count != 1)
            finaloutput += std::to_string(count); // increasing write-pointer
    }

    // writing to input
    for(int i = 0; i<finaloutput.size(); ++i){
        chars[i] = finaloutput[i];
    }

    // printing the final output
    cout << format("finaloutput = {}\n", finaloutput);
    cout << "chars = "; fPrintVector(chars);
    cout << format("return-value = {}\n", finaloutput.size());

    // return
```

```
    return(0);

}
```

530. Minimum Absolute Difference in BST

Given the root of a Binary Search Tree (BST), return the minimum absolute difference between the values of any two different nodes in the tree.

Constraints

- The number of nodes in the tree is in the range $[2, 10^4]$.
- $0 \leq \text{Node.val} \leq 105$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

    // input- configuration
    auto root          {new TreeNode(4)};
    root->left          = new TreeNode(2);
    root->right         = new TreeNode(6);
    root->left->left    = new TreeNode(1);
    root->left->right = new TreeNode(3);

    // setup
    vector<int> nodevalues;
    std::function<void(const TreeNode*)> foo = [&foo, &nodevalues](
        const TreeNode* root){
        // returning
```

```cpp
        if (root == nullptr) return;

        // going down left
        if (root->left)   {foo(root->left);}

        // adding current-value
        nodevalues.push_back(root->val);

        // going down right
        if (root->right)  {foo(root->right);}

        // returning
        return;
    };

    // calling function
    foo(root);

    // moving through node values
    auto minvalue {std::numeric_limits<int>::max()};
    auto temp     {-1};
    for(int i = 0; i<nodevalues.size()-1; ++i){
        // checking difference
        temp      = std::abs(nodevalues[i]- nodevalues[i+1]);
        minvalue  = std::min(minvalue, temp);
    }

    // returning minvalue
    cout << format("final-output = {}\n", minvalue);

    // return
    return(0);

}
```