Leetcode Solutions

SVR

August 31, 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;
```

```cpp
56        }
57  }
58
59  template<typename T, typename T1>
60  void fPrintHashmap(unordered_map<T, T1> input){
61      for(auto x: input){
62          cout << format("[{},{}] | ", x.first, x.second);
63      }
64      cout <<endl;
65  }
66
67  struct TreeNode {
68      int val;
69      TreeNode *left;
70      TreeNode *right;
71      TreeNode() : val(0), left(nullptr), right(nullptr) {}
72      TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
73      TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
74  };
75
76
77  struct ListNode {
78      int val;
79      ListNode *next;
80      ListNode() : val(0), next(nullptr) {}
81      ListNode(int x) : val(x), next(nullptr) {}
82      ListNode(int x, ListNode *next) : val(x), next(next) {}
83  };
84
85  void fPrintBinaryTree(TreeNode* root){
86      // sending it back
87      if (root == nullptr) return;
88
89      // printing
90      PRINTLINE
```

```cpp
        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
126      std::chrono::time_point<std::chrono::high_resolution_clock> endpoint;
127      std::chrono::duration<long long, std::nano>                 duration;
128
129      // constructor
130      Timer()      {startpoint = std::chrono::high_resolution_clock::now();}
131      void start() {startpoint = std::chrono::high_resolution_clock::now();}
132      void stop()  {endpoint = std::chrono::high_resolution_clock::now(); fetchtime();}
133
134      void fetchtime(){
135          duration = std::chrono::duration_cast<std::chrono::nanoseconds>(endpoint - startpoint);
136          cout << format("{} nanoseconds \n", duration.count());
137      }
138      void fetchtime(string stringarg){
139          duration = std::chrono::duration_cast<std::chrono::nanoseconds>(endpoint - startpoint);
140          cout << format("{} took {} nanoseconds \n", stringarg, duration.count());
141      }
142      void measure(){
143          auto temp = std::chrono::high_resolution_clock::now();
144          auto nsduration = std::chrono::duration_cast<std::chrono::nanoseconds>(temp - startpoint);
145          auto msduration = std::chrono::duration_cast<std::chrono::microseconds>(temp - startpoint);
146          auto sduration = std::chrono::duration_cast<std::chrono::seconds>(temp - startpoint);
147          cout << format("{} nanoseconds | {} microseconds | {} seconds \n",
148              nsduration.count(), msduration.count(), sduration.count());
149      }
150      ~Timer(){
151          measure();
152      }
153 };
154
155 // main-file =====================================================================
156 int main(){
157
158      // starting timer
159      Timer timer;
160
```

11

```
161        // input- configuration
162
163
164        // setup
165
166
167
168
169
170        // return
171        return(0);
172
173    }
```

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:

- $2 \leq$ 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
1   int main(){
2
3       // input- configuration
4       vector<int> nums  {2, 7, 11, 15};
5       int        target {9};
6
7       // setup
8       int                   complement    {0};
9       unordered_map<int, int> number_to_index;
10      vector<int>           finaloutput;
11
12      // filling the unordered_map
13      for(int i = 0; i < nums.size(); ++i){
14
15          // calculating complement
16          complement = target - nums[i];
17
18          // checking if complement is present in registry
19          if(number_to_index.find(complement) != number_to_index.end()) [[unlikely]]
20          {
21              finaloutput.push_back(number_to_index[complement]); // adding first index
22              finaloutput.push_back(i);                           // adding second index
23              break;                                              // breaking out
24          }
25          else [[likely]]
```

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

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

```
1   int main(){
2
3       // input- configuration
4       ListNode* l1  = new ListNode(2);
5       l1->next      = new ListNode(4);
6       l1->next->next = new ListNode(3);
7
8       ListNode* l2  = new ListNode(5);
9       l2->next      = new ListNode(6);
10      l2->next->next = new ListNode(4);
11
12      // setup
13      ListNode* traveller_1 = l1;
14      ListNode* traveller_2 = l2;
15      ListNode* finalOutput = new ListNode(-1);
16      ListNode* traveller_fo = finalOutput;
17
18      int sum                {0};
19      int carry              {0};
20      int value_1            {0};
21      int value_2            {0};
22
23      // moving through the two nodes
24      while(traveller_1 != nullptr || traveller_2 != nullptr){
25
26          // adding the two numbers
27          value_1 = traveller_1 == nullptr ? 0 : traveller_1->val;
```

```
28          value_2 = traveller_2 == nullptr ? 0 : traveller_2->val;
29
30          // calculating sum
31          sum    = value_1 + value_2 + carry;
32          if (sum >= 10) [[unlikely]]  {sum -= 10; carry = 1;}
33          else           [[likely]]    {carry = 0;}
34
35          // creating node
36          traveller_fo->next = new ListNode(sum);
37          traveller_fo       = traveller_fo->next;
38
39          // updating the two pointers
40          if(traveller_1 != nullptr) [[likely]] {traveller_1 = traveller_1->next;}
41          if(traveller_2 != nullptr) [[likely]] {traveller_2 = traveller_2->next;}
42      }
43
44      // creating a final node if carry is non-zero
45      if (carry == 1) [[unlikely]] {
46          traveller_fo->next = new ListNode(carry);
47      }
48
49      // printing the final output
50      traveller_fo = finalOutput->next;
51      cout << format("final-output = ");
52      while(traveller_fo != nullptr){
53          cout << traveller_fo->val << ", ";
54          traveller_fo = traveller_fo->next;
55      }
56      cout << "\n";
57
58      // return
59      return(0);
60
61  }
```

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

```
1    int main(){
2
3        // input- configuration
4        string s   {"tmmzuxt"};
5
6        // setup
7        unordered_map<char, int> histogram;
8        int p1 {0};
9        char curr;
10       int finaloutput {-1};
11       int temp_length {-1};
12
13       // going through the thing
14       for(int p2 = 0; p2<s.size(); ++p2){
15
16           // moving to another variable
17           curr = s[p2];
18
19           // checking if current character is in histogram
20           if (histogram.find(curr) == histogram.end()) [[unlikely]]
21           {
22               histogram[curr] = 1;
23           }
24           else [[likely]]
25           {
26               // checking if count is zero
27               if (histogram[curr] == 0)
28               {
29                   histogram[curr] = 1;
30               }
31               else
32               {
33                   // moving p1 until it arrives at first instance of curr
34                   while(s[p1] != curr)
```

```
35                    {
36                        --histogram[s[p1]];
37                        ++p1;
38                    }
39                    ++p1;
40                    histogram[curr] = 1;
41                }
42            }
43
44            // calculating longest length
45            finaloutput = finaloutput > (p2-p1+1) ? finaloutput : (p2-p1+1);
46        }
47
48        // printing
49        cout << format("longest length = {} \n", finaloutput);
50
51        // return
52        return(0);
53    }
```

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. $0 \le n \le 1000$

5. $1 \le m + n \le 2000$

6. $-10^6 \le$ nums1[i], nums2[i] $\le 10^6$

Code

```
1   int main(){
2
3       // input- configuration
4       vector<int> nums1 {1, 2};
5       vector<int> nums2 {3, 4};
6
7
8       // setup
9       vector<int>& first = nums1[0] <= nums2[0] ? nums1 : nums2;
10      vector<int>& second = nums1[0] > nums2[0] ? nums1 : nums2;
11      int left_first {0};
12      int right_first {static_cast<int>(first.size())-1};
13      int left_second {0};
14      int right_second  {static_cast<int>(second.size())-1};
15      int left_value = first[left_first] < second[left_second] ? first[left_first] : second[left_second];
16      int right_value = first[right_first] > second[right_second] ? first[right_first] : second[right_second];
17      int numiterations {static_cast<int>((nums1.size() + nums2.size())/2)};
18
19
20      // running for a certain number of iterations
21      for(int i = 0; i<numiterations+1; ++i){
22
23          // updating left
24          if (first[left_first] < second[left_second]) {left_value = first[left_first]; ++left_first;}
25          else                                          {left_value = second[left_second]; ++left_second;}
```

```
26         if (first[right_first] > second[right_second]) {right_value = first[right_first]; --right_first;}
27         else                                            {right_value = second[right_second]; --right_second;}
28
29         // printing
30         cout << format("left-value = {}, right-value = {}\n", left_value, right_value);
31     }
32
33     cout << format("median = {}\n", static_cast<double>(left_value + right_value)/2.0);
34
35
36
37     // return
38     return(0);
39
40 }
```

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. $1 \leq s.length \leq 1000$

2. s consists of English letters (lower-case and upper-case), ',' and '.'.

3. $1 \leq numRows \leq 1000$

Code

```cpp
1   int main(){
2
3       // input- configuration
4       string s   {"PAYPALISHIRING"};
5       int numRows {4};
6
7       // trivial case
8       if (numRows == 1) {cout << format("finaloutput = {}\n", s); return 0;}
9
10      // setup
11      int modlength {2*numRows-2};
12      int numblocks {(static_cast<int>(s.size())+ modlength-1)/modlength};
13      int sourceindex {-1};
14      string finaloutput;
15
16      // going through the thing
17      for(int row = 0; row < numRows; ++row){
18          for(int i = 0; i<numblocks; ++i){
19
20              // first column of each block
21              sourceindex = row + modlength * i;
22              if (sourceindex<s.size())        {finaloutput += s[sourceindex];}
23
```

```
24              // continuing in case of boundary rows
25              if (row == 0 || row == numRows-1) {continue;}
26
27              // taking care of the case where non-boundary rows
28              sourceindex = modlength - row + modlength*i;
29              if (sourceindex < s.size())        {finaloutput += s[sourceindex];}
30          }
31      }
32
33      // printing the final output
34      cout << format("final-output = {}\n", finaloutput);
35
36
37      // return
38      return(0);
39
40  }
```

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

```
1   int main(){
2
3       // input- configuration
4       vector<int> height {1,8,6,2,5,4,8,3,7};
5
6       // setup
7       int left      {0};
8       int right     {static_cast<int>(height.size())-1};
9       int maxvolume {-1};
10      int currvolume {-1};
11
12      // two-pointer approach
13      while(left < right){
14
15          // calculating volumes
16          currvolume = (right - left) * std::min(height[left], height[right]);
17          maxvolume  = maxvolume > currvolume ? maxvolume : currvolume;
18
19          // adjusting left and right based on volume
20          if (height[left] < height[right]) {++left;}
21          else                              {--right;}
22      }
23
24      // printing
25      cout << format("maxvolume = {}\n", maxvolume);
26
27      // return
28      return(0);
29
30  }
```

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

- $1 \leq num \leq 3999$

Code

```
1   int main(){
2
3       // input- configuration
4       int num    {1994};
5
6       // setup
7       vector<pair<int, string>> numToString {
8           {1, "I"},
9           {4, "IV"},
10          {5, "V"},
11          {9, "IX"},
12          {10, "X"},
13          {40, "XL"},
14          {50, "L"},
15          {90, "XC"},
16          {100, "C"},
17          {400, "CD"},
18          {500, "D"},
19          {900, "CM"},
20          {1000, "M"}
21      };                                                      // number-string pairs
22      string finaloutput;                                     // variable to hold the final output
23      int     count;                                          // variable that will hold the counts
24      auto mulstring = [](const int& count,
25                          const string& inputstring,
26                          string& finaloutput){
27          if (count == 0) {return;}
28          for(int i = 0; i<count; ++i){finaloutput += inputstring;}
29      };                                                      // lambda-function for int * string
            multiplications (python style)
30
31      // going through the hashmap from the end
32      for(int i = numToString.size()-1; i>=0; --i){
33
```

```
34          // calculating count
35          count   = num / numToString[i].first;
36          num     = num - numToString[i].first*count;
37
38          // adding to final output
39          mulstring(count, numToString[i].second, finaloutput);
40      }
41
42      // printing the final-output
43      cout << format("finaloutput = {}\n", finaloutput);
44
45      // return
46      return(0);
47
48  }
```

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. $1 \leq s.length \leq 15$

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

```cpp
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},
```

```
13                                              {'D', 500},
14                                              {'M', 1000}};
15
16      // going through the string
17      for(int i = 0; i<s.size(); ++i){
18          if ((i+1)<s.size() && charToInt[s[i]] < charToInt[s[i+1]]) {finaloutput -= charToInt[s[i]];}
19          else                                                        {finaloutput += charToInt[s[i]];}
20      }
21
22      // printing the final output
23      cout << format("finaloutput = {}\n", finaloutput);
24
25      // return
26      return(0);
27
28  }
```

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;
    }
```

```cpp
34
35      // subsetting and printing the prefix
36      prefix = string(strs[0].begin(), strs[0].begin()+p-1);
37      cout << format("finaloutput = {}\n", prefix);
38
39      // return
40      return(0);
41
42  }
```

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.
- $1 \leq sz \leq 30$
- $0 \leq Node.val \leq 100$
- $1 \leq n \leq sz$

Code

```
1  int main(){
2
3      // starting timer
4      Timer timer;
5
6      // input- configuration
7      ListNode* head                = new ListNode(1);
8      head->next                    = new ListNode(2);
9      head->next->next              = new ListNode(3);
10     head->next->next->next        = new ListNode(4);
11     head->next->next->next->next = new ListNode(5);
12     auto n  {2};
13
14     // trivial case
15     if (!head) {cout << format("final-output = "); fPrintLinkedList(head); cout << endl;}
16
17     // setup
18     auto nodecounter  {0};
19     auto prehead      {new ListNode};
20
21     prehead->next            = head;
22     ListNode* traveller      = head;
23     ListNode* delayedTraveller = prehead;
24
25     // going through the list
26     while(traveller){
27         if (++nodecounter > n) {delayedTraveller = delayedTraveller->next;}
28         traveller = traveller->next;
29     }
30
31     // reconnecting
32     delayedTraveller->next = delayedTraveller->next->next;
33
```

```
34      // sending back
35      cout << format("final-output = ");
36      fPrintLinkedList(prehead->next); cout << endl;
37
38
39
40
41
42      // return
43      return(0);
44
45   }
```

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

- $1 \le \text{s.length} \le 10^4$

- s consists of parentheses only '()[]'.

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       string s   {"()[]{}"};
8
9       // seutp
10      deque<char> var00;
11
12      // going through the string
13      for(auto x: s){
14          // pushing or poppping
15          if (x == '(' || x == '[' || x == '{') {var00.push_back(x);}
16          else{
17              if (var00.size()==0)              {cout << format("final-output = false\n"); return 0;}
18              if (x == ')' && var00.back() == '(') {var00.pop_back(); continue;}
19              if (x == ']' && var00.back() == '[') {var00.pop_back(); continue;}
20              if (x == '}' && var00.back() == '{') {var00.pop_back(); continue;}
21              cout << format("final-output = false\n"); return 0;
22          }
23      }
24
25      // checking if anything is left
```

```
26    if (var00.size()!=0) {cout << format("final-output = false\n"); return 0;}
27
28
29    // true-case
30    cout << format("final-output = true\n"); return 0;
31
32    // return
33    return(0);
34
35 }
```

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].

- $-100 \leq \text{Node.val} \leq 100$

- Both list1 and list2 are sorted in non-decreasing order.

Code

```cpp
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
```

```
28
29          if (traveller_1->next != nullptr)
30              traveller_1              = traveller_1->next;      // updating traveller 1
31          else{
32              traveller_1 = nullptr;
33              break;
34          }
35
36      }
37      else {
38          traveller_final->next = traveller_2;            // linking node to final
39          traveller_final       = traveller_final->next; // moving node to new link
40          if (traveller_2->next != nullptr)
41              traveller_2              = traveller_2->next;      // updating travel 2
42          else{
43              traveller_2 = nullptr;
44              break;
45          }
46      }
47  }
48
49  // checking if anything is left
50  while (traveller_1 != nullptr){
51      traveller_final->next = traveller_1;
52      traveller_final       = traveller_final->next;
53      traveller_1           = traveller_1->next;
54  }
55
56  while (traveller_2 != nullptr){
57      traveller_final->next = traveller_2;
58      traveller_final       = traveller_final->next;
59      traveller_2           = traveller_2->next;
60  }
61
62  // printing the final-output
```

```
63      cout << format("finaloutput = ");
64      fPrintLinkedList(finalOutput); cout << endl;
65
66      // return
67      return(0);
68  }
```

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){
```

```
6
7      // sending it back
8      if (numopens > n || numcloses > n) return;
9      else if (numopens == n && numcloses == n) {finalOutput.push_back(runningstring); return;}
10
11     // opening route
12     foo(numopens+1, numcloses, n, runningstring + "(", finalOutput);
13
14     // closing route
15     if (numopens>numcloses)
16         foo(numopens, numcloses+1, n, runningstring+")", finalOutput);
17
18     // returning
19     return;
20
21 }
22
23 int main(){
24
25     // starting timer
26     Timer timer;
27
28     // input- configuration
29     auto n {3};
30
31     // trivial case
32     if (n == 0) {cout << format("finalOutput = []\n"); return 0;}
33
34     // calling the function
35     vector<string> finalOutput;
36     foo(0, 0, n, "", finalOutput);
37
38     // returning
39     cout << format("finalOutput = {}\n", finalOutput);
40
```

```
41        // return
42        return(0);
43
44    }
```

23. Merge k Sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Examples

1. **Example 1:**

    - Input: lists = [[1,4,5],[1,3,4],[2,6]]
    - Output: [1,1,2,3,4,4,5,6]
    - Explanation: The linked-lists are:
      1→ 4→5,
      1→3→4,
      2→6.
      - merging them into one sorted linked list: 1→1→2→3→4→4→5→6

2. **Example 2:**

    - Input: lists = []
    - Output: []

3. **Example 3:**

    - Input: lists = [[]]
    - Output: []

Constraints

- k == lists.length

- $0 \le k \le 104$

- $0 \le$ lists[i].length ≤ 500

- $-10^4 \le$ lists[i][j] $\le 10^4$

- lists[i] is sorted in ascending order.

- The sum of lists[i].length will not exceed 10^4.

Code

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

    // starting timer
    Timer timer;

    // input- configuration


    // setup




    // return
    return(0);
```

```
18
19  }
```

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.
- $1 \leq k \leq n \leq 5000$
- $0 \leq Node.val \leq 1000$

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       ListNode* head                = new ListNode(1);
8       head->next                    = new ListNode(2);
9       head->next->next              = new ListNode(3);
10      head->next->next->next        = new ListNode(4);
11      head->next->next->next->next  = new ListNode(5);
12      auto k       {2};
13
14      // setup
15      ListNode* prehead = new ListNode(); prehead->next = head;
16      ListNode* traveller = prehead;
17      vector<ListNode*> memberList(k, nullptr); memberList.reserve(k);
18
19      // runnings
20      while(traveller){
21          // filling up the memberList
22          int counter {0};
23          ListNode* tempotraveller = traveller->next;
24          while(counter < k && tempotraveller != nullptr){
25              memberList[counter]  = tempotraveller;
26              tempotraveller        = tempotraveller->next;
27              ++counter;
28          }
29
30          // checking breaking condition
31          if (counter!=k) break;
32
33          // reconnecting
```

```
34        ListNode* beginningOfNextSegment = memberList[memberList.size()-1]->next;
35
36        // reconnecting
37        traveller->next = memberList[memberList.size()-1];
38        for(int i = memberList.size()-1; i>=1; --i) {memberList[i]->next = memberList[i-1];}
39        memberList[0]->next = beginningOfNextSegment;
40
41        // updating traveller
42        traveller = memberList[0];
43    }
44
45    // returning
46    cout << format("final-output = ");
47    fPrintLinkedList(prehead->next); cout << endl;
48
49    // return
50    return(0);
51
52 }
```

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);
```

```
24      cout << format("nums = "); fpv(nums);
25
26      // return
27      return(0);
28
29  }
```

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

- $0 \leq$ nums.length ≤ 100

- $0 \leq$ nums[i] ≤ 50

- $0 \leq$ val ≤ 100

Code

```
1   int main(){
2
3       // input- configuration
4       vector<int> nums  {0,1,2,2,3,0,4,2};
5       int val          {2};
6
7       // setup
8       int src       {0};
9       int dest      {0};
10      int numwrites {0};
11
12      // going through the indices
13      while(src < nums.size()){
14
15          // moving the dest until we find a val-position
16          while(nums[dest] != val) {++dest;}
17
18          // moving source until we find a non-val position after dest
19          src = std::max(src, dest+1);
20          while(nums[src] == val)  {++src;};
21
22          // writing
23          if (dest < nums.size() && src < nums.size()){
```

```
24              nums[dest] = nums[src];
25              ++dest;
26              ++src;
27              ++numwrites;
28          }
29
30      }
31
32      // printing the length
33      cout << format("updated nums = "); fPrintVector(nums);
34      cout << format("finaloutput = {} \n", nums.size()-numwrites-1);
35
36      // return
37      return(0);
38
39  }
```

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

```
1   int main(){
2
3       // input- configuration
4       string haystack {"leetcode"};
5       string needle {"leeto"};
6
7
8       // setup
9       int finaloutput {-1};
10      auto beginsearch = [haystack, needle](int currindex){
11          // starting search
12          if(currindex + needle.size() > haystack.size()) {return false;}
13
14          // checking if they're a subset
15          for(int i = 0; i<needle.size(); ++i){
16              if (haystack[currindex + i] != needle[i]) {return false;}
17          }
18
19          return true;
20      };
21
22      // going through
23      for(int i = 0; i < haystack.size(); ++i){
24
25          // begin search at each index
26          auto curroutput = beginsearch(i);
27
28          // writing final output, if a mach
29          if (curroutput) {finaloutput = i; break;}
30
31      }
32
33      // printing final output
```

66

```
34     cout << format("final-output = {}\n", finaloutput);
35
36     // return
37     return(0);
38
39 }
```

35. Search Insert Position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with $O(\log n)$ runtime complexity.

Examples

1. **Example 1:**

   - Input: nums = [1,3,5,6], target = 5
   - Output: 2

2. **Example 2:**

   - Input: nums = [1,3,5,6], target = 2
   - Output: 1

3. **Example 3:**

   - Input: nums = [1,3,5,6], target = 7
   - Output: 4

Constraints

- $1 \leq$ nums.length $\leq 10^4$

- $-10^4 \leq \text{nums[i]} \leq 10^4$

- nums contains distinct values sorted in ascending order.

- $-10^4 \leq \text{target} \leq 10^4$

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       auto nums  {vector<int>{1, 3, 5, 6}};
8       auto target {5};
9
10      // setup
11      auto left         {0};
12      auto right        {static_cast<int>(nums.size())-1};
13      auto mid          {-1};
14      auto finaloutput  {-1};
15
16      // running the loop
17      while(left<=right){
18
19          // updating mid
20          mid = (left +right)/2;
21
22          // checking if mid is the value
23          if (nums[mid] == target) {finaloutput = mid; cout << format("final-output = {}\n", finaloutput); return 0;}
24          else if(nums[mid]<target) {left = mid+1;}
25          else                      {right = mid-1;}
```

```
26
27        }
28
29        // returning the midvalue
30        finaloutput = left;
31        cout << format("final-output = {}\n", finaloutput);
32
33        // return
34        return(0);
35
36    }
```

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);

```

```
34          // going through each jumn
35          for(int j = 0; j<9; ++j){
36
37              // along the row
38              int var00  = fConvert(board[i][j]);
39              if (var00  != -1) {if (++rowRegister[var00-1] > 1) return false;}
40
41              // down the column
42              var00      = fConvert(board[j][i]);
43              if (var00  !=-1)  {if (++colRegister[var00-1] > 1) return false;}
44
45              // checking block
46              row_local  = j / 3;
47              col_local  = j % 3;
48              xoffset    = i / 3;
49              yoffset    = i % 3;
50
51              // calculating registers
52              var00 = fConvert(board[3*xoffset + row_local][col_local+3*yoffset]);
53              if (var00!=-1)     {if (++blockRegister[var00-1]>1) return false;}
54          }
55      }
56
57      // returning true
58      return true;
59  }
```

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. $0 \leq height[i] \leq 10^5$

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
```

76

```cpp
34      for(int i = 0; i < height.size(); ++i){
35
36          // finding max-height of the current column
37          auto minheight    = std::min({leftmaxes[i], rightmaxes[i]});   // finding max-height of borders
38          auto columnheight = minheight - height[i];                     // subtracting to find space
39          columnheight      = columnheight > 0 ? columnheight : 0;        // in case curr-height > max-height
40          finaloutput      += columnheight;                              // accumlating to water content
41      }
42
43      // printing the final output
44      cout << format("finaloutput = {}\n", finaloutput);
45
46      // return
47      return(0);
48
49  }
```

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:

- $0 \leq j \leq nums[i]$

- $i + j \leq n$

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

- $1 \leq$ nums.length $\leq 10^4$

- $0 \leq$ nums[i] ≤ 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
```

```
24          rightboundary = rightboundary < nums.size()-1 ?
25                          rightboundary : nums.size()-1;              // ensuring within vector range
26
27          // calculating smallest element in range
28          auto it = std::min_element(minjumps.begin()+leftboundary,
29                                 minjumps.begin()+rightboundary+1);    // finding the minimum value in the range
30
31          // addding min-element to the array
32          if (*it == std::numeric_limits<int>::max())
33              minjumps[i] = std::numeric_limits<int>::max();          // ensuring infty logic
34          else
35              minjumps[i] = (1 + *it);                                // for regular values
36
37      }
38
39      // printing
40      cout << format("finaloutput = {}\n", minjumps[0]);
41      timer.measure();
42
43      // return
44      return(0);
45
46  }
```

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

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       vector<vector<int>> matrix {
8           {5,1,9,11},
9           {2,4,8,10},
10          {13,3,6,7},
11          {15,14,12,16}
12      };
13
14
15      // setup
16      int t_edge = 0;
17      int b_edge = matrix.size()-1;
18      int l_edge = 0;
19      int r_edge = matrix[0].size() -1;
20      int temp;
21
22      // shifting layer by layer.
23      int numlayers = (matrix.size())/2;
24
25      // shifting layer by layer
26      for(int i = 0; i<numlayers; ++i){
27
28          // breaking if they're the same for some reason
29          if(t_edge == b_edge || l_edge == r_edge) break;
30
31          // calculatin width
32          int currwidth = r_edge - l_edge + 1;
33          for(int j = 0; j<currwidth-1; ++j){
```

```
34
35              // shifting the four elements for each
36              temp                      = matrix[t_edge][l_edge+j];
37              matrix[t_edge][l_edge+j] = matrix[b_edge-j][l_edge];
38              matrix[b_edge-j][l_edge] = matrix[b_edge][r_edge-j];
39              matrix[b_edge][r_edge-j] = matrix[t_edge+j][r_edge];
40              matrix[t_edge+j][r_edge]  = temp;
41          }
42
43          // updating edge-parameters based on the layer we're working with
44          t_edge += 1;
45          b_edge -= 1;
46          l_edge += 1;
47          r_edge -= 1;
48
49      }
50
51      // printing the matrix
52      cout << "final-matrix = \n";
53      for(const auto& x: matrix){
54          for(const auto& y: x){
55              cout << format("{}, ", y);
56          }
57          cout << format("\n");
58      }
59
60      // return
61      return(0);
62
63  }
```

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

- $1 \leq$ strs.length $\leq 10^4$

- $0 \leq$ strs[i].length ≤ 100

- strs[i] consists of lowercase English letters.

Code

```cpp
1   // build histogram
2   vector<int> fBuildHist(string input){
3       vector<int> myhist(26, 0);
4       for(auto x: input) {++myhist[static_cast<int>(x - 'a')];}
5       return myhist;
6   }
7
8   // Custom hash function for vector<int>
9   struct VectorHash {
10      std::size_t operator()(const std::vector<int>& v) const {
11          std::size_t hashValue = 0;
12          for (int num : v)
13              hashValue ^= std::hash<int>{}(num) + 0x9e3779b9 + (hashValue << 6) + (hashValue >> 2);
14
15          return hashValue;
16      }
17  };
18  // int main
19  int main(){
20
21      // starting timer
22      Timer timer;
23
```

```cpp
        // 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);
```

```
59    }
```

53. Maximum Subarray

Given an integer array nums, find the subarray with the largest sum, and return its sum.

Examples

1. **Example 1:**

   - Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
   - Output: 6
   - Explanation: The subarray [4,-1,2,1] has the largest sum 6.

2. **Example 2:**

   - Input: nums = [1]
   - Output: 1
   - Explanation: The subarray [1] has the largest sum 1.

3. **Example 3:**

   - Input: nums = [5,4,-1,7,8]
   - Output: 23
   - Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.

Constraints

- $1 \leq \text{nums.length} \leq 10^5$

- $-10^4 \leq \text{nums[i]} \leq 10^4$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

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

    // setup
    auto runningsum {0};
    auto finaloutput {std::numeric_limits<int>::min()};

    // going through the array
    for(int i =0; i<nums.size(); ++i){
        runningsum += nums[i];
        finaloutput = std::max(runningsum, finaloutput);
        if (runningsum <0) {runningsum = 0;}
    }

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

    // return
    return(0);

```

```
26    }
```

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

```cpp
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]);}
```

```cpp
34
35         // updating boundaries
36         ++left; --right; ++top; --bottom;
37
38     }
39
40     // printing the finaloutput
41     cout << format("final-output = {}\n", finalOutput);
42
43     // return
44     return(0);
45
46 }
```

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

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

Code

```cpp
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);
```

```
34
35  }
```

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

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

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       vector<vector<int>> intervals{
8           {1,3},
9           {2,6},
10          {8,10},
11          {15,18}
12      };
13
14      // setup
15      vector<vector<int>> finalOutput;
16
17      // sorting the input
18      std::sort(intervals.begin(), intervals.end(),
19          [](const vector<int>& a, const vector<int>& b){return a[0] < b[0];});
20
21      // going through the intervals
22      vector<int> runninginterval {intervals[0][0], intervals[0][1]};
23      for(int i = 1; i<intervals.size(); ++i){
24          if (runninginterval[1] < intervals[i][0]){
25              finalOutput.push_back(runninginterval);                        // pushing the results
26              runninginterval = intervals[i];                               // getting new running interfval
27          }
28          else    {runninginterval[1] = max(runninginterval[1], intervals[i][1]);}
29      }
30      finalOutput.push_back(runninginterval);                               // pushing the results
31
32      // returning
33      cout << format("final-output = {}\n", finalOutput);
```

```
34
35      // return
36      return(0);
37  }
```

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

- $0 \le$ intervals.length $\le 10^4$

- 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

```cpp
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
22      std::sort(intervals.begin(),
23                intervals.end(),
24                [](const vector<int>& a,
25                   const vector<int>& b){return a[0] < b[0];});
26      vector<int>        runninginterval {intervals[0]};
27
28      // going through the inputs
29      for(int i = 1; i<intervals.size(); ++i){
30          if (runninginterval[1] < intervals[i][0]){
31              finalOutput.push_back(runninginterval);
32              runninginterval = intervals[i];
33          }
34          else{
35              runninginterval[1] = max(runninginterval[1], intervals[i][1]);
36          }
37      }
38
39      // pushing it back
40      finalOutput.push_back(runninginterval);
41      cout << format("final-output = {}\n", finalOutput);
42
43      // return
44      return(0);
45
46  }
```

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

- $1 \leq s.length \leq 10^4$

- s consists of only English letters and spaces ''.

- There will be at least one word in s.

Code

```
1   int main(){
2
3       // input- configuration
4       string s    {"  fly me   to    the moon "};
5
6       // setup
7       int p1          {-1};
8       int finaloutput {-1};
9       string laststring;
10
11      // moving from the end
12      for(int i = s.size()-1; i>=0; --i){
13
14          // continuing until you find a non-space character
15          if (s[i] == ' ') {continue;}
16
17          // launch the start of first word
18          p1 = i;
19
20          // moving p1 until we find the first space or nonword thing
21          while(p1>=0 && s[p1]!=' ') {--p1;}
22
23          // calculating the length
```

```cpp
24        finaloutput = i - p1;
25        laststring = string(s.begin() + p1, s.begin() + i+1);
26
27        // breaking
28        break;
29     }
30
31     // printing
32     cout << format("length = {}, last-word = {}\n", finaloutput, laststring);
33
34     // return
35     return(0);
36
37  }
```

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
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       auto head  {new ListNode(1)};
8       head->next = new ListNode(2);
9       head->next->next = new ListNode(3);
10      head->next->next->next = new ListNode(4);
11      head->next->next->next->next = new ListNode(5);
12      auto k  {2};
13
14      // trivial cases
15      if (head == nullptr) {fPrintLinkedList("head = ", head); return 0;}
16
17      // storing the starting point
18      auto startingpoint {head};
19      auto listsize      {0};
20
21      // connecting the last-node with the first
22      auto traveller {head};
23      while(traveller->next) {traveller = traveller->next; ++listsize;}
24      ++listsize;
25      traveller->next = head;
26
27      // to get rid of cycles
28      k = k % listsize;
29      auto cutoffpoint  {listsize - k};
30      auto count        {0};
31      auto finalresult  {static_cast<ListNode*>(nullptr)};
32
33      // finding the cutting off point
```

```
34      traveller = head;
35      while(traveller){
36          ++count;                                              // appending count
37          if (count == cutoffpoint){
38              finalresult = traveller->next;                    // setting up the new head
39              traveller->next = nullptr;                        // setting up last pointer
40              break;                                            // breaking up
41          }
42          traveller = traveller->next;                          // moving to the next point
43      }
44
45      // printing the linked list
46      fPrintLinkedList("final-result = ", finalresult);
47
48      // return
49      return(0);
50
51  }
```

67. Add Binary

Given two binary strings a and b, return their sum as a binary string.

Examples

1. **Example 1:**

   - Input: a = "11", b = "1"
   - Output: "100"

2. **Example 2:**

   - Input: a = "1010", b = "1011"
   - Output: "10101"

Constraints

- $1 \leq$ a.length, b.length ≤ 104

- a and b consist only of '0' or '1' characters.

- Each string does not contain leading zeros except for the zero itself.

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       auto a {string("1010")};
8       auto b {string("1011")};
9
10
11      // setup
12      int carry {0};
13      int a_top {0};
14      int b_top {0};
15      int c_top {0};
16      string finalOutput {};
17
18      // running until the things are empty
19      while(a.size()!=0 || b.size()!=0){
20
21          // copying the carry over tot he sum
22          c_top = carry;
23
24          // shaving off the top
25          if (a.size()){
26              a_top  = static_cast<int>(a[a.size()-1] - '0');   // taking the top
27              a      = a.substr(0, a.size()-1);                 // removing the top
28              c_top  += a_top;                                   // adding to the sum
29          }
30          if (b.size()){
31              b_top  = static_cast<int>(b[b.size()-1] - '0');   // taking the top
32              b      = b.substr(0, b.size()-1);                 // removing the top
33              c_top  += b_top;                                   // adding to the sum
```

110

```
34          }
35
36          // evaluating carry
37          if (c_top == 0)          {carry = 0; c_top = 0;}
38          else if (c_top == 1)     {carry = 0; c_top = 1;}
39          else if (c_top == 2)     {carry = 1; c_top = 0;}
40          else if (c_top == 3)     {carry = 1; c_top = 1;}
41          else                     {cout << format("what the actual fuck") << endl;}
42
43          // adding c-TOp to the end of the final output
44          finalOutput = std::string(1,c_top+'0') + finalOutput;
45
46      }
47
48      // check if there is a carry out there
49      if (carry)    {finalOutput = std::string(1,carry+'0') + finalOutput;}
50
51      // returnign the final output
52      cout << format("final-output = {}\n", finalOutput);
53
54      // return
55      return(0);
56
57  }
```

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 "
     -]

113

Constraints

- $1 \le$ words.length ≤ 300

- $1 \le$ words[i].length ≤ 20

- words[i] consists of only English letters and symbols.

- $1 \le$ maxWidth ≤ 100

- words[i].length $\le$ 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 << ",";
```

```cpp
20      std::cout << std::endl;
21 }
22
23 // main-file ============================================================
24 int main(){
25
26     // input- configuration
27     auto words    = vector<string>({"This", "is", "an", "example", "of", "text", "justification."});
28     auto maxWidth {16};
29
30         // setup
31     std::vector<std::string> finalOutput;
32
33     // going through strings
34     int acc = 0;
35     int numwords = 0;
36     int currwidth = 0;
37     std::vector<std::string> temp;
38
39     for(int i = 0; i<words.size(); ++i){
40
41         // updating temp
42         temp.push_back(words[i]);    // updating words in temp
43
44         // checking if width has been crossed
45         if (fCalcLengthOfTempWithSpaces(temp) >= maxWidth || i == words.size()-1){
46
47             // condition temp based on length
48             if(fCalcLengthOfTempWithSpaces(temp)>maxWidth){
49                 temp.pop_back();      // last words gotta go
50                 --i;                  // making sure its taken care of in next iteration
51             }
52
53             // finding length of characters in temp
54             int num_nonspaces = fCalcLengthOfTempWithoutSpaces(temp);
```

115

```cpp
55
56            // finding number of spaces to add
57            int numspacestofill = maxWidth - num_nonspaces;
58
59            // calculating numspots
60            int numspots = temp.size()-1;
61
62            // calculating ideal number of spaces
63            int idealnumspacesperspot;
64            int remainders;
65            if (numspots!=0){
66                idealnumspacesperspot = numspacestofill/numspots;
67                remainders            = numspacestofill%numspots;
68            }
69            else{
70                idealnumspacesperspot = numspacestofill/1;
71                remainders            = numspacestofill%1;
72            }
73
74            // constructing candidate string
75            std::string candidate;
76
77            // adding each word in temp to the candidate
78            for(int j = 0; j < temp.size(); ++j){
79
80                // fetching word
81                auto x = temp[j];
82
83                // adding word to candidate
84                candidate += x;
85
86                // adding spaces
87                if (j!=temp.size()-1)
88                    for(int var00 = 0; var00 < idealnumspacesperspot; ++var00)
89                        candidate += " ";
```

```cpp
                    // 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
```

```
125        finalOutput[finalOutput.size()-1] = lastline;
126
127        // return
128        return(0);
129
130    }
```

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

- $1 \leq$ path.length ≤ 3000

- path consists of English letters, digits, period '.', slash '/' or '_'.

- path is a valid absolute Unix path.

Code

```cpp
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;}
    }
```

```cpp
34      // reconstructing final path
35      string finalOutput;
36
37      // reconstructing
38      for(auto x: stack) {finalOutput += "/" + x;}
39
40      // size thing
41      if (finalOutput.size() == 0) {finalOutput += "/";}
42
43      // returning
44      cout << format("finalOutput = {}\n", finalOutput);
45
46      // return
47      return(0);
48
49  }
```

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

```cpp
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");
```

```
34          }
35
36          // return
37          return(0);
38
39  }
```

74. Search a 2D Matrix

You are given an m x n integer matrix matrix with the following two properties:

1. Each row is sorted in non-decreasing order.

2. The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true if target is in matrix or false otherwise.

You must write a solution in O(log(m * n)) time complexity.

Examples

1. **Example 1**

   - Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
   - Output: true

2. **Example 2**

   - Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13
   - Output: false

Constraints

- m == matrix.length

- n == matrix[i].length

- $1 \le m, n \le 100$

- $-10^4 \le$ matrix[i][j], target $\le 10^4$

Code

```
1   struct Matrix{
2       vector<vector<int>> matrix;
3       int                 numcols;
4       int                 numrows;
5       Matrix(vector<vector<int>> matrix0): matrix(matrix0){
6           numcols = matrix0[0].size();
7           numrows = matrix0.size();
8       }
9       int operator()(int row, int col) {return matrix[row][col];}
10      int operator[](int tid)          {return matrix[tid/numcols][tid%numcols];}
11      int numel()                      {return matrix.size() * matrix[0].size();}
12  };
13
14  int main(){
15
16      // starting timer
17      Timer timer;
18
19      // input- configuration
20      vector<vector<int>> matrix = {
21          {1,3,5,7},
22          {10,11,16,20},
23          {23,30,34,60}};
24      auto target {3};
25
```

127

```cpp
26        // setup
27        Matrix m(matrix);
28        int left    {0};
29        int right   {static_cast<int>(matrix.size() * matrix[0].size()) - 1};
30        int mid     {-1};
31
32        // binary search
33        while(left <= right){
34            mid = (left + right)/2;
35            if (m[mid] < target)      {left = mid+1;}
36            else if (m[mid] > target) {right = mid-1;}
37            else                      {cout << format("final-output = true\n"); return 0;}
38        }
39
40        // returning false
41        cout << format("final-output = false\n");
42
43        // return
44        return(0);
45
46    }
```

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(){
```

```
34        // starting timer
35        Timer timer;
36
37        // input- configuration
38        auto n {4};
39        auto k {2};
40
41        // setup
42        vector< vector<int> > finalOutput;
43        vector<int>          pathsofar;
44
45        // calling the function
46        foo(pathsofar, n, k, finalOutput);
47
48        // printing the output
49        cout << format("finalOutput = {}\n", finalOutput);
50
51        // return
52        return(0);
53
54    }
```

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};

```

```cpp
     6      // setup
     7      int destination    {1};
     8      int prev           {nums[0]};
     9      int element_counter {1};
    10      int numwrites      {1};
    11
    12      // going through the values
    13      for(int i = 1; i < nums.size(); ++i){
    14
    15          // updating counter
    16          if (nums[i-1] == nums[i]) {++element_counter;}
    17          else                      {element_counter = 1;}
    18
    19          // checking the element counters
    20          if (element_counter <=2) {nums[destination++] = nums[i];}
    21
    22      }
    23
    24      // printing the final output
    25      cout << format("nums = "); fpv(nums);
    26      cout << format("return-value = {}\n", destination);
    27
    28      // return
    29      return(0);
    30
    31  }
```

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].
- $-100 \leq Node.val \leq 100$
- $-200 \leq x \leq 200$

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       auto head  {new ListNode(1)};
8       head->next = new ListNode(4);
9       head->next->next = new ListNode(3);
10      head->next->next->next = new ListNode(2);
11      head->next->next->next->next = new ListNode(5);
12      head->next->next->next->next->next = new ListNode(2);
13      auto x  {3};
14
15      // setup
16      ListNode* traveller      = head;
17      ListNode* lefthead       = new ListNode(-1);
18      ListNode* lefttraveller  = lefthead;
19      ListNode* righthead      = new ListNode(-1);
20      ListNode* righttraveller = righthead;
21
22      // going through the list
23      while(traveller){
24          // checking next-value
25          if (traveller->val < x){
26              lefttraveller->next   = new ListNode(traveller->val);
27              lefttraveller         = lefttraveller->next;
28          }
29          else {
30              righttraveller->next = new ListNode(traveller->val);
31              righttraveller       = righttraveller->next;
32          }
33
```

135

```
34        // moving traveller
35        traveller = traveller->next;
36    }
37
38    // connecting
39    lefttraveller->next = righthead->next;
40
41    // returning
42    fPrintLinkedList("Final-output = ", lefthead->next);
43
44    // return
45    return(0);
46
47 }
```

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. $0 \leq m, n \leq 200$

4. $1 \leq m + n \leq 200$

5. $-10^9 \leq nums1[i], nums2[j] \leq 10^9$

Code

```cpp
1   int main(){
2
3       // input- configuration
4       vector<int> nums1 {1, 2, 3, 0, 0, 0};
5       vector<int> nums2 {2, 5, 6};
6       int m   {3};
7       int n   {3};
8
9       // setup
10      int p1      {m-1};
11      int p2      {n-1};
12      int p3      {m+n-1};
```

```
13
14     int curr1  {-1};
15     int curr2  {-1};
16
17     // going the other way
18     while(p1 >= 0 || p2 >= 0)
19     {
20         // printing the values
21         curr1 = p1 >= 0 ? nums1[p1] : std::numeric_limits<int>::min();
22         curr2 = p2 >= 0 ? nums2[p2] : std::numeric_limits<int>::min();
23
24         // assigning value
25         if (curr1 > curr2) {nums1[p3] = curr1; --p3; --p1;}
26         else               {nums1[p3] = curr2; --p3; --p2;}
27
28     }
29
30     // printing the final output
31     cout << format("finaloutput = "); fPrintVector(nums1);
32
33     // return
34     return(0);
35 }
```

92. Reverse Linked List II

Given the head of a singly linked list and two integers left and right where left i= 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.

- $1 \leq n \leq 500$

- $-500 \leq$ Node.val ≤ 500

- $1 \leq$ left $\leq$ right $\leq n$

Code

```
1  int main(){
2
3      // starting timer
4      Timer timer;
5
6      // input- configuration
7      ListNode* head                 = new ListNode(1);
8      head->next                     = new ListNode(2);
9      head->next->next               = new ListNode(3);
10     head->next->next->next         = new ListNode(4);
11     head->next->next->next->next   = new ListNode(5);
12     auto left  {2};
13     auto right {4};
14
15     // trivial case
16     if (left == right) {fPrintLinkedList(head); cout << endl; return 0;}
17
18     // setup
19     ListNode* traveller  = nullptr;
20     ListNode* entrypoint = nullptr;
21     ListNode* exitpoint  = nullptr;
22     ListNode* revhead    = nullptr;
23     ListNode* dummy      = new ListNode(0);
24     dummy->next          = head;
25
26     // moving through list
27     int nodeposition = -1;
28     traveller = dummy;
29     while(traveller){
30
31         ++nodeposition;    // updating node position
32
33         // finding important points
```

```
34        if(nodeposition == left-1)       {entrypoint = traveller;}
35        else if(nodeposition == left)    {revhead   = traveller;}
36        else if(nodeposition == right+1) {exitpoint = traveller;}
37
38        // moving traveller to next point
39        traveller = traveller->next;
40    }
41
42    // reversing the head
43    ListNode* prev        = nullptr;
44    ListNode* actualright = nullptr;
45    traveller             = revhead;
46    nodeposition          = left;
47
48    while(traveller && nodeposition <= right){
49        ++nodeposition;
50        actualright    = traveller->next; // storing original right
51        traveller->next = prev;           // reconnection
52        prev           = traveller;       // storing prev for next iteration
53        traveller      = actualright;     // moving to original right
54    }
55
56    // tying things together
57    entrypoint->next  = prev;
58    revhead->next     = exitpoint;
59
60    // returning dummy-es nstex
61    fPrintLinkedList(dummy->next); cout << endl;
62
63    // return
64    return(0);
65
66 }
```

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

```
int main(){

    // starting timer
    Timer timer;

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

```

```cpp
        // 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
```

```cpp
46      cout << format("final-output = {}\n", validBSTFlag);
47
48      // return
49      return(0);
50  }
```

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].
- $-10^4 \leq$ 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);
```

```
34
35     // return
36     return(0);
37
38 }
```

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 ≤ 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));
```

```
35
36     // return
37     return(0);
38  }
```

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

```cpp
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
```

```cpp
20          if (root == nullptr) return;
21
22          // finding current-level
23          level += 1;
24
25          // increasing size of the thing
26          while (finalOutput.size()<level+1)
27              finalOutput.push_back(std::vector<int>());
28
29          // appending to value
30          if (finalOutput.size() < level+1)
31              finalOutput[level] = std::vector<int>(root->val);
32          else
33              finalOutput[level].push_back(root->val);
34
35          // calling
36          f(root->left, level);
37          f(root->right, level);
38
39          // returning return;
40          return;
41
42      };
43
44      // going through the tree
45      f(root, -1);
46
47      // printing the final output
48      cout << format("finalOutput = {}\n", finalOutput);
49
50      // return
51      return(0);
52
53  }
```

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

- $1 \leq$ nums.length $\leq 10^4$

- $-10^4 \leq$ nums[i] $\leq 10^4$

- nums is sorted in a strictly increasing order.

Code

```
1   void foo(TreeNode*      parent,
2            int            parentdirection,
3            vector<int>&   nums,
4            int            left,
5            int            right)
6   {
7       // sending it back if right is less than left
8       if (left > right) return;
9
10      // finding mid point
11      int mid = (left + right)/2;
12
13      // creating mid-node
14      TreeNode* midnode = new TreeNode(nums[mid]);
15
16      // connecting current to parent node
```

```
17      if (parentdirection == -1)        {parent->right = midnode;}
18      else if (parentdirection == 1)    {parent->left  = midnode;}
19
20      // going left
21      foo(midnode, 1, nums, left, mid-1);
22
23      // going right
24      foo(midnode, -1, nums, mid+1, right);
25
26      // returning
27      return;
28
29  }
30  int main(){
31
32      // starting timer
33      Timer timer;
34
35      // input- configuration
36      vector<int> nums  {-10,-3,0,5,9};
37
38      // setup
39      TreeNode* dummy = new TreeNode(-1);
40
41      // calling the function
42      foo(dummy, -1, nums, 0, nums.size()-1);
43
44      // return
45      return(0);
46
47  }
```

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 ≤ 1000

- $-1000 \leq$ targetSum ≤ 1000

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       auto root                {new TreeNode(5)};
8
9       root->left              = new TreeNode(4);
10      root->right             = new TreeNode(8);
11
12      root->left->left        = new TreeNode(11);
13      root->right->left       = new TreeNode(13);
14      root->right->right      = new TreeNode(4);
15
```

```cpp
16        root->left->left->left   = new TreeNode(7);
17        root->left->left->right  = new TreeNode(2);
18        root->right->right->right = new TreeNode(1);
19
20        auto targetSum     {22};
21
22        // setup
23        auto runningsum    {0};
24        auto binaryflag    {false};
25        std::function<void(const TreeNode*, int)> fRunSum = [&fRunSum,
26                         &targetSum,
27                         &binaryflag](
28            const TreeNode* root,
29            int         runningsum
30            ){
31
32            // sending it back
33            if (root       == nullptr)    return;
34            if (binaryflag == true)       return;
35
36            // adding current val to running sum
37            runningsum += root->val;
38
39            // checking if there are children
40            if (root->left || root->right){
41                if (root->left)   fRunSum(root->left,   runningsum);
42                if (root->right)  fRunSum(root->right,  runningsum);
43            }
44            else{
45                // in this case, we check the running sum
46                if (runningsum == targetSum){ binaryflag = true;}
47            }
48
49            // returning
50            return;
```

```
51        };
52
53        // running
54        fRunSum(root, runningsum);
55
56        // printing
57        cout << format("final-output = {}\n", binaryflag);
58
59        // return
60        return(0);
61
62 }
```

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].

- $-100 \leq$ Node.val ≤ 100

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       auto root          {new TreeNode(1)};
8       root->left         = new TreeNode(2);
9       root->right        = new TreeNode(5);
10      root->left->left   = new TreeNode(3);
11      root->left->right  = new TreeNode(4);
12      root->right->right = new TreeNode(6);
13
14      // recursion lambda
15      auto foo = [&foo](TreeNode*         root,
16                        vector<TreeNode*>&  nodeaddresses){
17
18          // sending it back
19          if (root == nullptr) return;
20
21          // adding address
22          nodeaddresses.push_back(root);
23
24          // going down left
25          if (root->left) foo(root->left, nodeaddresses);
```

```cpp
            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

```
1  struct Node {
2      int val;
3      Node *left;
4      Node *right;
5      Node *next;
6  }
```

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

```cpp
1  class Node {
2  public:
3      int val;
4      Node* left;
5      Node* right;
6      Node* next;
7
8      Node() : val(0), left(NULL), right(NULL), next(NULL) {}
9      Node(int _val) : val(_val), left(NULL), right(NULL), next(NULL) {}
10     Node(int _val, Node* _left, Node* _right, Node* _next)
11         : val(_val), left(_left), right(_right), next(_next) {}
12 };
13
14
15 int main(){
16
17     // starting timer
18     Timer timer;
19
20     // input- configuration
21     auto root         {new Node(1)};
22     root->left        = new Node(2);
23     root->right       = new Node(3);
24     root->left->left  = new Node(4);
25     root->left->right = new Node(5);
```

163

```
26      root->right->right = new Node(7);
27
28      // running
29      if (root == nullptr) {cout << format("done!\n"); return 0;}
30
31      // calling the function
32      int count = -1;
33      vector< vector<Node*> > addressvectors;
34
35      // calling th efunction
36      foo(root, count, addressvectors);
37
38      // building the connections
39      for(auto x: addressvectors){
40          int i = 0;
41          for(; i<x.size()-1; ++i)
42              if (x[i] != nullptr) {x[i]->next = x[i+1];}                    // look to the right
43
44          // ensuring that the last pointer points to thr right most
45          x[i]->next = nullptr;
46      }
47
48      // completion
49      cout << format("done!\n");
50
51      // return
52      return(0);
53
54  }
```

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:

- $1 \leq$ prices.length $\leq 10^5$
- $0 \leq$ 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

- $1 \leq$ prices.length $\leq 3 * 10^4$

- $0 \leq$ prices[i] $\leq 10^4$

Code

```
1   int main(){
2
3       // input- configuration
4       vector<int> prices {7,1,5,3,6,4};
5
6       // setup
7       int p1        {0};                              // index-pointer to buying
8       int p2        {0};                              // index-pointer to selling
9       int accprofit {0};                              // variable to accumulate profit
10      int currprofit {std::numeric_limits<int>::min()}; // variable to hold curr-profit
11
12      // going through this
13      while(p2 < prices.size()){
14
15          currprofit = prices[p2] - prices[p1];       // calculating current profit
16
17          if (currprofit > 0){
18              accprofit  += currprofit;               // accumulating the profit
19              p1          = p2++;                      // moving the starting point
20              continue;                               // moving into the next iteration
21          }
22          else if (currprofit < 0){
23              p1          = p2++;                      // moving the starting point
24              continue;
25          }
```

```cpp
26
27          ++p2;                                                    // updating p2
28      }
29
30      // printing the max-value
31      cout << format("accprofit = {}\n", accprofit);
32
33      // return
34      return(0);
35
36  }
```

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 ≤ 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
15      auto finalresult  {std::numeric_limits<int>::min()};
16
17      // creating lambda
18      std::function<int(const TreeNode*)> fFindMaxPathSum = [&fFindMaxPathSum,
19                                                             &finalresult](
20          const TreeNode* root){

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

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

28              // the potential from leaf-node
29              return root->val;
30          }
31          else{

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

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

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

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

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

```
50
51            // path from left-curr-right
52            potentials.push_back(leftplusright + root->val);
53
54            // auto temp = *(std::max_element(potentials.begin(), potentials.end()));
55            const auto comparitivemax = *(std::max_element(potentials.begin(), potentials.end()));
56
57            // finalresult = std::max(finalresult, maxelement);
58            finalresult = finalresult > comparitivemax ? finalresult : comparitivemax;
59
60            // returning the max-potential from here
61            return maxelement;
62        }
63    };
64
65
66    // calculating the final-result
67    fFindMaxPathSum(root);
68
69    // printing
70    cout << format("final-result = {}\n", finalresult);
71
72    // return
73    return(0);
74
75 }
```

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

- $1 \leq$ s.length $\leq 2 * 10^5$

- s consists only of printable ASCII characters.

Code

```cpp
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;}

```

```cpp
26          // checking bounds
27          if (pleft>=pright) {break;}
28
29          // checking if they're the same
30          if (std::tolower(s[pleft]) != std::tolower(s[pright])) {finaloutput = false; break;}
31
32          // updating pointers
33          ++pleft; --pright;
34      }
35
36      // printing
37      cout << format("final-output = {}\n", finaloutput);
38
39      // return
40      return(0);
41
42  }
```

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

```cpp
// 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()) {
```

```
16          cout << format("final-output = 0\n");
17          return 0;
18      }
19
20      // setup
21      unordered_map<string, std::set<string> > neighbours;
22
23      auto calculatedistances = [](const string a, const string b){
24          auto numdiffs {0};
25          for(int i = 0; i<a.size(); ++i) {if (a[i] != b[i]) {++numdiffs;}}
26          if (numdiffs == 0)          {return 0;}
27          else if (numdiffs == 1)     {return 1;}
28          else                        {return std::numeric_limits<int>::max();}
29      };
30
31      // starting again
32      vector<string> candidates {beginWord};
33      for(const auto x: wordList) {if (x!=endWord) candidates.push_back(x);}
34      vector<string> nextgencandidates {};
35
36      vector<string> recruiters        {{endWord}};
37      vector<string> nextgenrecruiters {};
38
39      // going through the loop
40      auto count  {1};
41      auto runningcondition {true};
42      auto foundpath        {false};
43
44      while(runningcondition){
45          // increasing count
46          ++count;
47
48          // comparing distance between candidates and recruiters
49          for(const auto candidate: candidates){
50
```

```
51              auto shortestpath {std::numeric_limits<int>::max()};
52              for(const auto& recruiter: recruiters)
53                  shortestpath  = std::min(shortestpath, calculatedistances(candidate, recruiter));
54
55              // adding to next-generation of recruiters if diff == 1
56              if (shortestpath == 1){
57                  nextgenrecruiters.push_back(candidate);      // adding to next gen
58                  if (candidate == beginWord) {foundpath = true; runningcondition = false;}
59              }
60              else if (shortestpath == std::numeric_limits<int>::max()){
61                  nextgencandidates.push_back(candidate);
62              }
63          }
64
65          // rewriting history
66          int prevnumcandidates = candidates.size();
67          int nexnumcandidates  = nextgencandidates.size();
68          candidates = nextgencandidates; nextgencandidates.clear();
69          recruiters = nextgenrecruiters; nextgenrecruiters.clear();
70
71          // evaluating running condition
72          if (prevnumcandidates == nexnumcandidates) {runningcondition = false;}
73
74      }
75
76      // setting up final output
77      if (!foundpath) {count = 0;}
78
79      // printing final-output
80      cout << format("final-output = {}\n", count);
81
82
83      // return
84      return(0);
85
```

```
86    }
```

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

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

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       auto nums = vector<int>({0,3,7,2,5,8,4,6,0,1});
8
9       // setup
10      std::multiset<int> var00;
11
12      // trivial-cases
13      if (nums.size() <= 1) {cout << format("final-output = {}\n", nums.size()); return 0;}
14
15      // going through elements
16      for(auto x: nums) {var00.insert(x);}
17
18      // going through elements
19      int maxlength = std::numeric_limits<int>::min();
20      int temp = 1;
21      std::deque<int> var01;
22
23      for(auto x: var00){
24
25          if (var01.size()<1)          {var01.push_back(x);}
26          else{
27              // comparing previous element and current
28              if (x - var01[0] == 1)   {++temp; var01[0] = x; maxlength = max(maxlength, temp);}
29              else if(x - var01[0] == 0) {maxlength = max(maxlength, temp); continue;}
30              else                     {maxlength = max(maxlength, temp); temp = 1; var01[0] = x;}
31          }
32      }
33
```

182

```
34      // returning the max-length
35      cout << format("final-output = {}\n", maxlength);
36
37      // return
38      return(0);
39
40  }
```

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].

- $0 \leq$ Node.val ≤ 9

- The depth of the tree will not exceed 10.

Code

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

15
16        // setup
17        auto runningsum {0};
18        auto finalSum {0};
19        std::function<void(const TreeNode*, int)> foo = [&foo,
20                                           &finalSum](
21            const TreeNode*   root,
22            int               runningsum){

23
24            // sending it back
25            if (root == nullptr) return;

26
27            // adding current value to running sum
28            runningsum = runningsum*10 + root->val;

29
30            // going down left or right
31            if (root->left || root->right){
32                if (root->left)   foo(root->left,   runningsum);
33                if (root->right)  foo(root->right,  runningsum);
34            }
35            else{
36                // adding to final sum
37                finalSum += runningsum;
38            }

39
40            // return re
41            return;
42        };

43
44        // calling function
45        foo(root, runningsum);

46
47        // retunring finalSum
48        cout << format("final-output = {}\n", finalSum);
```

```
49
50      // return
51      return(0);
52
53  }
```

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

```cpp
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);
```

188

```
34
35    }
```

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. $1 \le n \le 2 * 10^4$

3. $0 \le ratings[i] \le 2 * 10^4$

Code

```cpp
// 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
```

```
24          if (currrating > leftrating){
25              currcount = leftcount+1;
26          }
27      }
28
29      // right pass
30      for(int i = ratings.size()-2; i>=0; --i){
31
32          // fetching ratings
33          currrating = ratings[i];
34          rightrating = ratings[i+1];
35
36          // fetching references to candies
37          int& currcandies  = candies[i];
38          int& rightcandies = candies[i+1];
39
40          // updating based on right
41          if (currrating > rightrating){
42              currcandies = std::max(currcandies,
43                                      rightcandies + 1);
44          }
45      }
46
47      // summing up candies
48      finaloutput = std::accumulate(candies.begin(), candies.end(), 0);
49      cout << format("finaloutput = {}\n", finaloutput);
50
51      // return
52      return(0);
53
54  }
```

192

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

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       ListNode* head                = new ListNode(3);
8       head->next                    = new ListNode(2);
9       head->next->next              = new ListNode(0);
10      head->next->next->next        = new ListNode(-4);
11      head->next->next->next->next = head->next;
12
13      // trivial case
14      if (head == nullptr) {cout << format("final-output = false\n"); return 0;}
15
16      // setup
17      ListNode* dummy   = new ListNode(0);
18      dummy->next       = head;
19      ListNode* fastboi = dummy;
20      ListNode* slowboi = dummy;
21
22      while (fastboi != nullptr && fastboi->next != nullptr) {
23
```

```
24          // updating positions
25          fastboi = fastboi->next->next;
26          slowboi = slowboi->next;
27
28          // checking if they're the same
29          if (slowboi == fastboi) {cout << format("final-output = true\n"); return 0;}
30      }
31
32      // since exiting the list implies you left
33      cout << format("final-output = false\n"); return 0;
34
35      // return
36      return(0);
37  }
```

148. Sort List

Given the head of a linked list, return the list after sorting it in ascending order.

Constraints

- The number of nodes in the list is in the range $[0, 5 * 10^4]$.

- $-105 \leq$ Node.val ≤ 105

Code

```cpp
int main(){

    // starting timer
    Timer timer;

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

    // setup
    if (head == nullptr)  {cout << format("empty-list\n"); return 0;}

    // setup
    ListNode* traveller = nullptr;
    vector<pair<int, ListNode*>> nodeaddresses;

    // goign through the list
```

```
20      traveller = head;
21      while(traveller){
22
23          // adding it ot the list
24          nodeaddresses.push_back(std::pair<int, ListNode*>({traveller->val, traveller}));
25
26          // moving on to the next one
27          traveller = traveller->next;
28      }
29
30      // sorting the whole thing
31      std::sort(nodeaddresses.begin(),
32                nodeaddresses.end(),
33                [](pair<int, ListNode*> a,
34                   pair<int, ListNode*> b) {return a < b;});
35
36      // reconnecting
37      for(int i = 0; i<nodeaddresses.size()-1; ++i)
38          nodeaddresses[i].second->next = nodeaddresses[i+1].second;
39
40
41      // making the last one connect to nullptr
42      nodeaddresses[nodeaddresses.size()-1].second->next = nullptr;
43
44      // returning top of the list
45      fPrintLinkedList("finalOutput = ", nodeaddresses[0].second);
46
47      // return
48      return(0);
49  }
```

150. Evaluate Reverse Polish Notation

You are given an array of strings tokens that represents an arithmetic expression in a Reverse Polish Notation.

Evaluate the expression. Return an integer that represents the value of the expression.

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. $1 \leq s.\text{length} \leq 10^4$

2. s contains English letters (upper-case and lower-case), digits, and spaces ' '.

3. There is at least one word in s.

Code

```
1   int main(){
2
3       // input- configuration
4       string s   {"a good  example"};
5
6       // setup
7       vector<string> listofwords;
8
9       // creating a list of words
10      int p1 {0};
11      string acc;
12      while(p1 < s.size()){
13
14          // checking if the current character is a non-space
15          if (s[p1] != ' ') {acc += s[p1];}
16          else{
17              // if acc is non-empty, flush
18              if (acc.size() != 0)  {listofwords.push_back(acc); acc = "";}
19              else                  {;}
20          }
21
22          // moving the index-pointer forward
23          p1++;
```

```
24          }
25
26          // check if acc is unflushed
27          if (acc.size() != 0)  {listofwords.push_back(acc); acc = "";}
28
29          // building the finaloutput
30          string finaloutput;
31          for(int i = listofwords.size()-1; i>=0; --i){
32              finaloutput += listofwords[i];
33              if (i!=0) [[unlikely]] {finaloutput += " ";}
34          }
35
36          // printing the finaloutput
37          cout << format("finaloutput = {}\n", finaloutput);
38
39
40          // return
41          return(0);
42
43      }
```

153. Find Minimum in Rotated Sorted Array

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

- [4,5,6,7,0,1,2] if it was rotated 4 times.

- [0,1,2,4,5,6,7] if it was rotated 7 times.

Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of unique elements, return the minimum element of this array.

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

Examples

1. **Example 1:**
    - Input: nums = [3,4,5,1,2]
    - Output: 1
    - Explanation: The original array was [1,2,3,4,5] rotated 3 times.

2. **Example 2:**
    - Input: nums = [4,5,6,7,0,1,2]
    - Output: 0
    - Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.

3. **Example 3:**

   - Input: nums = [11,13,15,17]
   - Output: 11
   - Explanation: The original array was [11,13,15,17] and it was rotated 4 times.

Constraints

- n == nums.length
- $1 \leq n \leq 5000$
- $-5000 \leq nums[i] \leq 5000$
- All the integers of nums are unique.
- nums is sorted and rotated between 1 and n times.

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       auto nums  {vector<int>{3,4,5,1,2}};
8
9       // setup
10      auto left  {0};
```

```cpp
11      auto right {static_cast<int>(nums.size())-1};
12      auto mid   {-1};
13
14      // looping through
15      while(left < right){
16
17          // calculating midpoint
18          mid = (left + right)/2;
19
20          // moving pointers
21          if(nums[mid] > nums[right])  {left  = mid + 1;}
22          else                         {right = mid;}
23      }
24
25      // returning the mid-value
26      cout << format("final-output = {}\n", nums[left]);
27
28      // return
29      return(0);
30
31  }
```

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] ≤ 1000

- numbers is sorted in non-decreasing order.

- $-1000 \leq$ target ≤ 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){
```

```cpp
15
16          // checking sum of two values
17          currsum = numbers[left] + numbers[right];
18
19          // comparing against target
20          if (currsum > target)     {--right;}
21          else if (currsum < target) {++left;}
22          else {
23              finalOutput.push_back( left+1);
24              finalOutput.push_back( right+1);
25              break;
26          }
27      }
28
29      // printign the final output
30      cout << format("finaloutput = {}\n", finalOutput);
31
32      // return
33      return(0);
34
35  }
```

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
- $1 \leq n \leq 5 * 10^4$
- $-10^9 \leq nums[i] \leq 10^9$

Code

```
1   int main(){
2
3       // input- configuration
4       vector<int> nums  {2,2,1,1,1,2,2};
5
6       // setup
7       unordered_map<int, int> histogram;
8       int max_element   {std::numeric_limits<int>::min()};
9       int max_count     {std::numeric_limits<int>::min()};
10      int updated_count {0};
11
12      // going through the elements
13      for(int i = 0; i<nums.size(); ++i){
14
15          // adding to histogram
16          if (histogram.find(nums[i]) == histogram.end()) {histogram[nums[i]] = 1; updated_count = 0;}
17          else                                            {++histogram[nums[i]]; updated_count = histogram[nums[i]];}
18
19          // keeping track of max-element
20          if (updated_count > max_count) {max_element = nums[i]; max_count = updated_count;}
21
22      }
23
24      // printing the final output
25      cout << format("nums = "); fpv(nums);
26      cout << format("max-count = {}\n", max_count);
27
28      // return
29      return(0);
30
31  }
```

209

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

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

Code

```
1  int main(){
2
3      // input- configuration
4      vector<int> nums  {-1,-100,3,99};
5      int k  {2};
6
7      // setup
8      StopWatch timer;                                      // setting up the timer
9      k = k %static_cast<int>(nums.size());                 // to ensure that the value is within range
10
11     int source         {0};
12     int temp_source    {nums[source]};
13     int temp           {0};
14     int destination    {0};
15
16     vector<bool> sourcelist(nums.size(), false);
17
18     // going through nums
19     for(int i = 0; i < nums.size(); ++i){
20
21         // check if curent-source has been taken care of
22         if (sourcelist[source] == true){
23             source      = (source+1) % nums.size();
24             temp_source = nums[source];
25         }
26
27         source             = source % nums.size();        // code to ensure range
28         destination        = (source + k)%nums.size();    // calculating the index we'll be writing to
29         sourcelist[source] = true;                        // updating source-list
30
31         temp               = nums[destination];           // safe-keeping the destination value
32         nums[destination] = temp_source;                  // storing new value at destination-index
33
```

```
34        source           = destination;                    // updating source-index
35        temp_source      = temp;                           // updating source-value
36    }
37
38    // printing the output
39    cout << format("nums = "); fpv(nums);                  // printing the updated array, "nums"
40    timer.stop();                                          // printing the time taken
41
42    // return
43    return(0);
44 }
```

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

- $1 \le m, n \le 300$

- grid[i][j] is '0' or '1'.

Code

```cpp
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]};
```

```
26
27          // checking validity
28          if (row >= 0          &&
29              row < grid.size() &&
30              col >= 0          &&
31              col <grid[0].size())
32              if (grid[row][col] == "1") {return true;}
33
34          // in case the above condition is not met
35          return false;
36
37      };
38
39      // traversal function
40      std::function<void(int, int)> fMarkThemAll = [&fMarkThemAll,
41                                                     &grid,
42                                                     &didWeVisitThisPlace,
43                                                     fCheckValidityOfCoordinate](
44          int row_index,
45          int col_index){
46
47          // setting up coordinates
48          std::vector<int> rightCoordinate({row_index, col_index+1});
49          std::vector<int> downCoordinate{row_index+1, col_index};
50
51          // marking the current coordinate as visited
52          didWeVisitThisPlace[row_index][col_index] = true;
53
54          // calling the function to the right
55          if (fCheckValidityOfCoordinate(rightCoordinate) == true &&\
56              didWeVisitThisPlace[rightCoordinate[0]][rightCoordinate[1]] == false)
57                  fMarkThemAll(rightCoordinate[0], rightCoordinate[1]);
58
59          // calling the function for the ones below
60          if (fCheckValidityOfCoordinate(downCoordinate) == true && \
```

215

```
61              didWeVisitThisPlace[downCoordinate[0]][downCoordinate[1]] == false)
62                  fMarkThemAll(downCoordinate[0], downCoordinate[1]);
63       };
64
65
66       // going through the elements
67       int count = 0;
68       for(int row_index = 0; row_index < grid.size(); ++row_index){
69           for(int col_index = 0; col_index < grid[0].size(); ++col_index){
70               // starting an exploratory course if this point has not been visited
71               if (didWeVisitThisPlace[row_index][col_index] == false && \
72                   grid[row_index][col_index]                == "1"){
73                   fMarkThemAll(row_index, col_index);
74                   ++count;
75               }
76           }
77       }
78
79       // return count
80       cout << format("final-output = {}\n", count);
81
82       // return
83       return(0);
84
85  }
```

216

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:
     - $1^2 + 9^2 = 82$
     - $8^2 + 2^2 = 68$
     - $6^2 + 8^2 = 100$
     - $1^2 + 0^2 + 0^2 = 1$

2. **Example 2:**
   - Input: n = 2
   - Output: false

Constraints

- $1 \le n \le 2^31 - 1$

Code

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       auto n {19};
8
9       // trivial case
10      if (n == 4 || n == 0) {cout << format("final-output = false\n"); return 0;}
11
12      // setting up lambda
13      auto fSumDigitSquares = [](int n) -> int
14      {
15          auto sum   {0};
16          auto digit {-1};
17          while(n!=0){
18              digit  = n%10;
19              sum    += digit*digit;
20              n      = n/10;
21          }
22          return sum;
23      };
24
25      // calling the function
26      while (n != 1){
27          // calculating digits sums
```

```
28          n = fSumDigitSquares(n); cout << format("n = {}\n", n);
29          if (n == 4 || n == 0)    {cout << format("final-output = false\n"); return 0;}
30      }
31
32      // printing and returning
33      cout << format("final-output = true \n"); return 0;
34
35  }
```

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:

- $1 \le s.length \le 5 * 10^4$

- 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
```

```cpp
                    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
```

```
59      cout << format("final-output = true \n");
60
61      // return
62      return(0);
63  }
```

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

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

- $1 \leq$ nums.length $\leq 10^5$

- $1 \leq$ nums[i] $\leq 10^4$

Code

```
1   int main(){
2
3       // input- configuration
4       vector<int> nums  {1,2,3,4,5};
5       auto target    {15};
6
7       // setup
8       auto finaloutput  {std::numeric_limits<int>::max()};
9       auto pleft        {0};
10      auto sum          {nums[pleft]};
11      auto i            {1};
12
13      // in case the first element itself is greater
14      if (sum > target) {finaloutput = 1;}
15
16      // lambda to update finaloutput
17      auto updatefinaloutput = [&sum,
18                                &target,
19                                &finaloutput,
20                                &i,
21                                &pleft,
22                                &nums]() -> void{
23
24          auto numelements = i - pleft + 1;
25          finaloutput = numelements < finaloutput ? numelements : finaloutput;
26          sum -= nums[pleft++];
27      };
```

```cpp
28
29        // going through the array
30        for(; i<nums.size(); ++i){
31
32            // adding to sum
33            sum += nums[i];
34
35            // updating
36            while (sum>=target) {updatefinaloutput();}
37        }
38
39        // updating
40        if(finaloutput == std::numeric_limits<int>::max()) {finaloutput = 0;}
41
42        // printing the finaloutput
43        cout << format("finaloutput = {}\n", finaloutput);
44
45        // return
46        return(0);
47
48    }
```

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

```
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;
```

228

```
8
9        // calculating sum so far
10       int sumsofar = std::accumulate(numberPath.begin(), numberPath.end(),0);
11
12       // number to fill
13       int sumtofill = target - sumsofar;
14
15       // if this is zero, we can add it to list and just sendn it back
16       if (sumtofill == 0){
17           if (numberPath.size() == k){
18               if (std::find(finalOutput.begin(),
19                             finalOutput.end(),
20                             numberPath) == finalOutput.end())
21                   finalOutput.push_back(numberPath);                    // add if it doesn't already exist
22           }
23
24           return;
25       }
26
27       // valid candidates
28       std::vector<int> candidates;
29       int biggest_number = 1;
30       if (numberPath.size()!=0){
31           auto iter = std::max_element(numberPath.begin(), numberPath.end());
32           biggest_number = *iter;
33       }
34
35       sumtofill = std::min(sumtofill, 9);
36       for(int i = biggest_number; i<=sumtofill; ++i){
37           // add if no on th epath so far
38           if (std::find(numberPath.begin(),
39                         numberPath.end(),
40                         i) == numberPath.end()) {candidates.push_back(i);}
41       }
42
```

```cpp
43      // if there are no candidates, we're going back
44      if (candidates.size() == 0) return;
45
46      // trying each candidate
47      std::vector<int> numberPath_local;
48      for(auto x: candidates){
49          // sending down each candidate route
50          numberPath_local = numberPath;
51          numberPath_local.push_back(x);
52          fTraverse(numberPath_local, target, k, finalOutput);
53      }
54
55      // returning
56      return;
57  }
58
59  int main(){
60
61      // starting timer
62      Timer timer;
63
64      // input- configuration
65      auto k {3};
66      auto n {7};
67
68      // setup
69      std::vector< std::vector<int> > finalOutput;
70      std::vector<int> numberPath;
71
72      // recursion
73      fTraverse(numberPath, n, k, finalOutput);
74
75      // printing
76      cout << format("final-output = {}\n", finalOutput);
77
```

```
78        // return
79        return(0);
80
81    }
```

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

- $1 \leq$ nums.length $\leq 10^5$
- $-10^9 \leq$ 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);

```

233

```
34    }
```

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

```cpp
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
```

236

```
34      fTwist(root);
35
36      // return
37      return(0);
38
39  }
```

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] ⟹ "0→2"
     - [4,5] ⟹ "4→5"
     - [7,7] ⟹ "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→4"
  - $[6,6] \implies$ "6"
  - $[8,9] \implies$ "8→9"

Constraints

- $0 \leq$ nums.length ≤ 20

- $-2^31 \leq$ nums[i] $\leq 2^31 - 1$

- All the values of nums are unique.

- nums is sorted in ascending order.

Code

```cpp
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
```

```cpp
13        auto arrow {"->"};
14        auto p1      {0};
15        auto p2      {0};
16        vector<string> finalOutput;
17
18        // while loop
19        while (p2<nums.size()){
20
21            // checking prev element
22            if (nums[p2] <= 1 + nums[p2-1]) {++p2;}
23            else{
24                if (p1!=p2-1) {finalOutput.push_back(std::to_string(nums[p1]) + arrow + std::to_string(nums[p2-1]));}
25                else          {finalOutput.push_back(std::to_string(nums[p2-1]));}
26                p1 = p2++;
27            }
28        }
29
30        // checking if p1 and p2 areright ne
31        if (p1!=p2-1) {finalOutput.push_back(std::to_string(nums[p1]) + arrow + std::to_string(nums[p2-1]));}
32        else          {finalOutput.push_back(std::to_string(nums[p2-1]));}
33
34        // returning restul
35        cout << format("final-output = {}\n", finalOutput);
36
37        // return
38        return(0);
39
40    }
```

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.

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

- $0 \leq Node.val \leq 10^4$

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};
```

```cpp
         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

```cpp
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];
    }

```

```
34      // printing
35      cout << format("finaloutput = "); fPrintVector(finaloutput);
36
37      // return
38      return(0);
39
40  }
```

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

```
1   int main(){
2
3       // input- configuration
4       vector<int> citations {3,0,6,1,5};
5
6       // sorting the citations first
7       std::sort(citations.begin(), citations.end(),
8           [](const int& a, const int& b) {return a>b;});
9
10      // running accumulations
11      auto hvalue    {0};
12      for(int i = 0; i<citations.size(); ++i){
13          if (citations[i] >= (i+1))    {hvalue = i+1;}
14      }
15
16      // printing citations
17      cout << format("hvalue = {}\n", hvalue);
18
19      // return
20      return(0);
21
22  }
```

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:

- $1 \leq nums.length \leq 10^4$
- $-2^{31} \leq nums[i] \leq 2^{31} - 1$

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

- $1 \leq s.length \leq 3 * 10^5$

- s consist of printable ASCII characters.

Code

```
1   int main(){
2
3       // input- configuration
4       string s   {"leetcode"};
5
6       // going through the string
7       string     vowels          {"aeiouAEIOU"};
8       vector<int> vowel_indices;
9       string     reversed_vowels;
10      string     finaloutput = s;
11
12      // going through the string
13      for(int i = 0; i<s.size(); ++i){
14          if (vowels.find(s[i]) != string::npos){
15              reversed_vowels+=s[i];
16              vowel_indices.push_back(i);
17          }
18      }
19
20      // refilling the indices
21      for(int i = 0; i<vowel_indices.size(); ++i){
22          finaloutput[vowel_indices[i]] = reversed_vowels[reversed_vowels.size()-1-i];
23      }
24
25      // printing the final output
26      cout << format("finaloutput = {}\n", finaloutput);
27
28      // return
29      return(0);
30
31  }
```

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

- $0 \leq s.length \leq 100$

- $0 \leq t.length \leq 10^4$

- s and t consist only of lowercase English letters.

Code

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

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

```
1   int main(){
2
3       // input- configuration
4       string s   {"100[leetcode]"};
5
6       // running
7       std::stack<char>  mystack;                                          // stack
8       string            temp;                                             // temporary string used for decoding
9       int               repcount   {1};                                  // used for decoding
10
11      // going through the inputs
12      for(int i = 0; i<s.size(); ++i){
13
14          if(s[i] != ']')   {mystack.push(s[i]);}                        // pushing characters to stack until we
                  arrive at "]"
15          else{
16              temp = "";                                                 // initializing temporary string
17              while(mystack.top() != '[') {                              // expanding mini-string until we arrive at
                      "["
18                  temp = mystack.top() + temp;
19                  mystack.pop();
```

```
20            }
21
22            mystack.pop();                                                      // removing "["
23
24            // calculating the repcount
25            string numberasstring = "";
26            while(mystack.size() != 0 &&
27                  mystack.top() - '0' >= 0 && '9' - mystack.top() >= 0)
28            {
29                numberasstring = mystack.top() + numberasstring;
30                mystack.pop();
31            }
32            repcount = std::stoi(numberasstring);
33
34            // mini-decoding
35            int multitempsize = repcount * temp.size();                          // calculating size after multiplication
36            for(int j = 0; j<multitempsize; ++j) {                               // filling up the stack with decoded content
37                mystack.push(temp[j%temp.size()]);
38            }
39        }
40    }
41
42    // creating the final output
43    string finaloutput;
44    while(mystack.size()){
45        finaloutput = mystack.top() + finaloutput;
46        mystack.pop();
47    }
48
49    // printing the final output
50    cout << format("finaloutput = {}\n", finaloutput);
51
52    // return
53    return(0);
54
```

```
55    }
```

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" → "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

- $0 \leq$ bank.length ≤ 10

- startGene.length $==$ endGene.length $==$ bank[i].length $== 8$

- startGene, endGene, and bank[i] consist of only the characters ['A', 'C', 'G', 'T'].

Code

```cpp
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;
```

```
24
25          // updating path sof far
26          auto pathsofar_temp = pathsofar;
27          pathsofar_temp.push_back(x);
28
29          // calling function do it
30          foo(stringtoneighbours, pathsofar_temp, foundpath, endGene, finaloutput);
31      }
32  }
33
34  // main-file =========================================================================
35  int main(){
36
37      // starting timer
38      Timer timer;
39
40      // input- configuration
41      string startGene      = "AACCGGTT";
42      string endGene        = "AACCGGTA";
43      vector<string> bank   = {"AACCGGTA"};
44
45      // setup
46      unordered_map<string, vector<string>> stringtoneighbours;
47
48      // checking if endgene is in the bank
49      if (std::find(bank.begin(),
50                    bank.end(),
51                    endGene) == bank.end()) {cout << format("finalOutput = -1\n");}
52
53      // going through the bank and building neighbours
54      bank.push_back(startGene);
55
56      for(auto x: bank){
57
58          // finding valid transactions with the other strings in the bank
```

```cpp
59          for(auto y: bank){

60
61              // checking number of string differences between the two
62              auto count = 0;
63              for(int i =0; i<8; ++i) {if (x[i] != y[i]) ++count;}

64
65              // checking count and adding to valid transactions
66              if (count == 1)        {stringtoneighbours[x].push_back(y);}
67          }

68
69          // checking if this particular gene can jump to final gene
70          // checking number of string differences between the two
71          auto count = 0;
72          for(int i =0; i<8; ++i) {if (x[i] != endGene[i]) ++count;}

73
74          // checking count and adding to valid transactions
75          if (count == 1)        {stringtoneighbours[x].push_back(endGene);}
76      }

77
78      // recursion
79      vector<string> pathsofar {startGene};
80      bool            foundpath  {false};
81      int             finalOutput {-1};

82
83      // calling the function
84      foo(stringtoneighbours, pathsofar, foundpath, endGene, finalOutput);

85
86      // returning the finaloutput
87      if (finalOutput > -1) --finalOutput;

88
89      // printing final output
90      cout << format("finalOutput = {}\n", finalOutput);

91
92      // return
93      return(0);
```

```
94
95   }
```

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

- $1 \leq$ chars.length ≤ 2000
- chars[i] is a lowercase English letter, uppercase English letter, digit, or symbol.

Code

```
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
```

```cpp
17            curr = chars[p1];
18
19            // increasing count
20            if (count == 0)              {runningchar = chars[p1]; ++count;}
21            else if(curr == runningchar) {++count;}
22            else if(curr != runningchar) {
23                finaloutput += runningchar; // writing character to current pointer
24                if (count != 1)
25                    finaloutput += std::to_string(count); // increasing write-pointer
26                runningchar = curr;
27                count = 1;
28            }
29
30            // increasing pointer
31            ++p1;
32        }
33
34        // flushing out
35        if (count != 0){
36            finaloutput += runningchar; // writing character to current pointer
37            if (count != 1)
38                finaloutput += std::to_string(count); // increasing write-pointer
39        }
40
41        // writing to input
42        for(int i = 0; i<finaloutput.size(); ++i){
43            chars[i] = finaloutput[i];
44        }
45
46        // printing the final output
47        cout << format("finaloutput = {}\n", finaloutput);
48        cout << "chars = "; fPrintVector(chars);
49        cout << format("return-value = {}\n", finaloutput.size());
50
51        // return
```

267

```
52    return(0);
53
54  }
```

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

```
1   int main(){
2
3       // starting timer
4       Timer timer;
5
6       // input- configuration
7       auto root          {new TreeNode(4)};
8       root->left        = new TreeNode(2);
9       root->right       = new TreeNode(6);
10      root->left->left  = new TreeNode(1);
11      root->left->right = new TreeNode(3);
12
13      // setup
14      vector<int> nodevalues;
15      std::function<void(const TreeNode*)> foo = [&foo, &nodevalues](
16          const TreeNode* root){
17          // returning
```

```
18        if (root == nullptr) return;
19
20        // going down left
21        if (root->left)    {foo(root->left);}
22
23        // adding current-value
24        nodevalues.push_back(root->val);
25
26        // going down right
27        if (root->right)   {foo(root->right);}
28
29        // returning
30        return;
31    };
32
33    // calling function
34    foo(root);
35
36    // moving through node values
37    auto minvalue {std::numeric_limits<int>::max()};
38    auto temp     {-1};
39    for(int i = 0; i<nodevalues.size()-1; ++i){
40        // checking difference
41        temp      = std::abs(nodevalues[i]- nodevalues[i+1]);
42        minvalue  = std::min(minvalue, temp);
43    }
44
45    // returning minvalue
46    cout << format("final-output = {}\n", minvalue);
47
48    // return
49    return(0);
50
51 }
```

918. Maximum Sum Circular Subarray

Given a circular integer array nums of length n, return the maximum possible sum of a non-empty subarray of nums.

A circular array means the end of the array connects to the beginning of the array. Formally, the next element of nums[i] is nums[(i + 1) % n] and the previous element of nums[i] is nums[(i - 1 + n) % n].

A subarray may only include each element of the fixed buffer nums at most once. Formally, for a subarray nums[i], nums[i + 1], ..., nums[j], there does not exist i ≤ k1, k2 ≤ j with k1 % n == k2 % n.

Examples

1. **Example 1:**

   - Input: nums = [1,-2,3,-2]
   - Output: 3
   - Explanation: Subarray [3] has maximum sum 3.

2. **Example 2:**

   - Input: nums = [5,-3,5]
   - Output: 10
   - Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10.

3. **Example 3:**

   - Input: nums = [-3,-2,-3]
   - Output: -2
   - Explanation: Subarray [-2] has maximum sum -2.

Constraints

- n == nums.length

- $1 \leq n \leq 3 * 10^4$

- $-3 * 10^4 \leq nums[i] \leq 3 * 10^4$

Code

```cpp
int main(){

    // starting timer
    Timer timer;

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

    // setup
    int global_maximum    {nums[0]};
    int global_minimum    {nums[0]};
    int running_max       {0};
    int running_min       {0};
    int total             {0};

    // going through the array
    for(auto x: nums){

        // calculating running maximum and minimum values
        running_max = max(running_max + x, x);
        running_min = min(running_min + x, x);

        // running up the total value
```

```
24          total += x;
25
26          // updating global maximum and minumum values
27          global_maximum = max(global_maximum, running_max);
28          global_minimum = min(global_minimum, running_min);
29      }
30
31      // some condition
32      if (global_maximum > 0)
33          global_maximum = max(global_maximum, total - global_minimum);
34
35      // printing the maximum
36      cout << format("final-output = {}\n", global_maximum);
37
38      // return
39      return(0);
40
41  }
```