Leetcode practice

Tuesday, June 4, 2024   11:01 AM

1. Two sum (Easy)

The Two Sum problem asks us to find two numbers in an array that sum up to a given target value. We need to return the indices of these two numbers.

Brute Force

```cpp
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int n = nums.size();
        for (int i = 0; i< n; i++){
            for (int j = i + 1; j < n; j++){
                if (nums[i] + nums[j] == target){
                    return {i, j};
                }
            }
        }
        return {};
    }
};
```

2. Roman to integer

The key intuition lies in the fact that in Roman numerals, when a smaller value appears before a larger value, it represents subtraction, while when a smaller value appears after or equal to a larger value, it represents addition.

```cpp
#include <unordered_map>
#include <iostream>
#include <string>
class Solution {
public:
    int romanToInt(std::string s) {
        std::unordered_map<char, int> m;
        // Define the values for each Roman numeral
        m['I'] = 1;
        m['V'] = 5;
        m['X'] = 10;
        m['L'] = 50;
        m['C'] = 100;
        m['D'] = 500;
        m['M'] = 1000;
        int ans = 0;
        // Iterate through the string
        for (int i = 0; i < s.length(); i++) {
            // Check if the current value is less than the next value
            if (i < s.length() - 1 && m[s[i]] < m[s[i + 1]]) {
                ans -= m[s[i]];
            } else {
                ans += m[s[i]];
            }
```

```cpp
        }
        return ans;
    }
};

int main() {
    Solution solution;
    std::string roman = "MCMXCIV"; // Example Roman numeral
    int result = solution.romanToInt(roman);
    std::cout << "The integer value of " << roman << " is: " << result
<< std::endl;
    return 0;
}
```

For the example "IX":
- When i is 0, the current character s[i] is 'I'. Since there is a next character ('X'), and the value of 'I' (1) is less than the value of 'X' (10), the condition m[s[i]] < m[s[i+1]] is true. In this case, we subtract the value of the current character from ans.
  ans -= m[s[i]];
  ans -= m['I'];
  ans -= 1;
  ans becomes -1.
- When i is 1, the current character s[i] is 'X'. This is the last character in the string, so there is no next character to compare. Since there is no next character, we don't need to evaluate the condition. In this case, we add the value of the current character to ans.
  ans += m[s[i]];
  ans += m['X'];
  ans += 10;
- ans becomes -1 +10 = 9

3. Palindrome number
The intuition behind this code is to reverse the entire input number and check if the reversed number is equal to the original number. If they are the same, then the number is a palindrome

```cpp
#include <iostream>
using namespace std;
int main() {
  int n, reversed_number = 0, remainder;
  cout << "Enter an integer: ";
  cin >> n;
  while(n != 0) {
    remainder = n % 10;
    reversed_number = reversed_number * 10 + remainder;
    n /= 10;
  }
  cout << "Reversed Number = " << reversed_number;
  return 0;
}
```

n	n != 0	remainder	reversed_number
2345	true	5	0 * 10 + 5 = 5
234	true	4	5 * 10 + 4 = 54
23	true	3	54 * 10 + 3 = 543
2	true	2	543 * 10 + 2 = 5432
0	false	-	Loop terminates.

4.Maxiumum Subarray

The Intuition behind the code is to find the maximum sum of a contiguous subarray within the given array nums. It does this by scanning through the array and keeping track of the current sum of the subarray. Whenever the current sum becomes greater than the maximum sum encountered so far, it updates the maximum sum. If the current sum becomes negative, it resets the sum to 0 and starts a new subarray. By the end of the loop, the code returns the maximum sum found.

From <https://leetcode.com/problems/maximum-subarray/solutions/3666304/beats-100-c-java-python-beginner-friendly/>

```cpp
#include <iostream>
#include <vector>
#include <climits>
class Solution {
public:
    int maxSubArray(std::vector<int>& nums) {
        int maxSum = INT_MIN;
        int currentSum = 0;
        for (int i = 0; i < nums.size(); i++) {
            currentSum += nums[i];
            if (currentSum > maxSum) {
                maxSum = currentSum;
            }

            if (currentSum < 0) {
                currentSum = 0;
            }
        }
        return maxSum;
    }
};
int main() {
    Solution solution;
    std::vector<int> nums = {-2, 1, -3, 4, -1, 2, 1, -5, 4}; // Example input
    int result = solution.maxSubArray(nums);
    std::cout << "The maximum subarray sum is " << result << std::endl;
```

```
        return 0;
}
```

Index	Array								Curr_Sum	Max_Sum
i=0	-2	-3	4	-1	-2	1	5	-3	-2	-2
i=1	-2	-3	4	-1	-2	1	5	-3	-3	-2
i=2	-2	-3	4	-1	-2	1	5	-3	4	4
i=3	-2	-3	4	-1	-2	1	5	-3	3	4
i=4	-2	-3	4	-1	-2	1	5	-3	1	4
i=5	-2	-3	4	-1	-2	1	5	-3	2	4
i=6	-2	-3	4	-1	-2	1	5	-3	7	7
i=7	-2	-3	4	-1	-2	1	5	-3	4	7

Array = [-2, -3, 4, -1, -2, 1, 5, -3]

5.Remove Element

The intuition behind this solution is to iterate through the array and keep track of two pointers: index and i. The index pointer represents the position where the next non-target element should be placed, while the i pointer iterates through the array elements. By overwriting the target elements with non-target elements, the solution effectively removes all occurrences of the target value from the array.

From <https://leetcode.com/problems/remove-element/solutions/3670940/best-100-c-java-python-beginner-friendly/>

```cpp
#include <iostream>
#include <vector>
#include <climits>
class Solution {
public:
    int removeElement(std::vector<int>& nums, int val) {
        // Represents the current position for the next non-target element
        int index = 0;
        for (int i = 0; i < nums.size(); i++) {
            // If nums[i] is not equal to val, it means it is a non-target
element
            if (nums[i] != val) {
                nums[index] = nums[i];
                index++;
            }
        }
        return index;
    }
}
```

```cpp
};
int main() {
    Solution solution;
    std::vector<int> nums = {0, 1, 2, 2, 3, 0, 4, 2}; // Initialize the
vector correctly
    int val = 2;
    int newLength = solution.removeElement(nums, val);

    std::cout << "The array after removing the element " << val << " is: ";
    for (int i = 0; i < newLength; i++) {
        std::cout << nums[i] << " ";
    }
    std::cout << std::endl;
    std::cout << "The new length of the array is " << newLength << std::endl;

    return 0;
}
```

6.contain duplicate

The hash set approach uses a hash set data structure to store encountered elements. It iterates through the array, checking if an element is already in the set. If so, it returns true. Otherwise, it adds the element to the set. This approach has a time complexity of O(n) and provides an efficient way to check for duplicates.

From <https://leetcode.com/problems/contains-duplicate/solutions/3672475/4-method-s-c-java-python-beginner-friendly/>

```cpp
#include <iostream>
#include <vector>
#include <unordered_set>

class Solution {
public:
    bool containsDuplicate(std::vector<int>& nums) {
        std::unordered_set<int> check_duplicate;
        for (int num : nums) {
            //std::unordered_set::find() function returns an iterator
pointing to  the element if it is found in the set,
            //and it returns std::unordered_set::end() if the element
is not found
            if (check_duplicate.find(num) != check_duplicate.end()) {
                return true;
            }
            check_duplicate.insert(num);
        }
        return false;
    }
};

int main() {
    Solution solution;
    std::vector<int> nums = {1, 2, 3, 4, 5, 6, 7, 8, 9, 1}; // Example
input
    bool result = solution.containsDuplicate(nums);
    if (result) {
```

```cpp
        std::cout << "The array contains duplicates." << std::endl;
    } else {
        std::cout << "The array does not contain duplicates."
<< std::endl;
    }
    return 0;
}
```