PROBLEM SOLVING

LeetCode

Two Sum

Problem Description:

https://leetcode.com/problems/two-sum/

Solution:

```
class Solution {
  List<int> twoSum(List<int> nums, int target) {
    // create a Map to store the complement of each number
    var numsMap = <int, int>{};

    // iterate through the list
    for (var i = 0; i < nums.length; i++) {
      // calculate the complement of the current element
      var complement = target - nums[i];

      // if the complement is already in the map, return the
indices
      if (numsMap.containsKey(complement)) {
        return [numsMap[complement]!, i];
      }

      // otherwise, add the current number and its index to the
map
      numsMap[nums[i]] = i;
    }

    // if no solution is found, return an empty list
    return [];
  }
}
```

Palindrome Number

Problem Description:

https://leetcode.com/problems/palindrome-number/

Solution:

```dart
class Solution {
  bool isPalindrome(int x) {
    return x.toString().split('').reversed.join() == x.toString();
  }
}
```

Roman to Integer

Problem Description:

https://leetcode.com/problems/roman-to-integer/description/

Solution:

```dart
class Solution {
  int romanToInt(String s) {
    Map<String, int> romanSymbols = {
      'I': 1,
      'V': 5,
      'X': 10,
      'L': 50,
      'C': 100,
      'D': 500,
      'M': 1000
    };

    int result = 0;

    for (int i = 0; i < s.length; i++) {
      if (i > 0 && romanSymbols[s[i]]! > romanSymbols[s[i - 1]]!)
{
        result += romanSymbols[s[i]]! - 2 * romanSymbols[s[i -
1]]!;
      } else {
        result += romanSymbols[s[i]]!;
      }
    }

    return result;
  }
}
```

Longest Common Prefix

Problem Description:

https://leetcode.com/problems/longest-common-prefix/description/

Solution:

```
class Solution {
  String longestCommonPrefix(List<String> strs) {
    if (strs.isEmpty)
      return '';
    else {
      String longestCommonPrefix = strs[0];

      for (int i = 1; i < strs.length; i++) {
        while (strs[i].indexOf(longestCommonPrefix) != 0) {
          // remove the last letter until indexOf == 1 (There's a
match)
          longestCommonPrefix =
              longestCommonPrefix.substring(0,
longestCommonPrefix.length - 1);

          // if no match & longestCommonPrefix become
          if (longestCommonPrefix.isEmpty) return '';
        }
      }
      return longestCommonPrefix;
    }
  }
}
```

Valid Parentheses

Problem Description:

https://leetcode.com/problems/valid-parentheses/

Solution:

```
class Solution {
  bool isValid(String string) {
```

```
    List<String> stack = <String>[];

    for (int i = 0; i < string.length; i++) {
      String bracket = string[i];

      if (bracket == '(' || bracket == '{' || bracket == '[') {
        stack.add(bracket);
      } else if (bracket == ')' && (stack.isEmpty || stack.last !=
'(')) {
        return false;
      } else if (bracket == '}' && (stack.isEmpty || stack.last !=
'{')) {
        return false;
      } else if (bracket == ']' && (stack.isEmpty || stack.last !=
'[')) {
        return false;
      } else {
        stack.removeLast();
      }
    }

    return stack.isEmpty;
  }
}
```

Merge two sorted lists

Problem Description:

https://leetcode.com/problems/merge-two-sorted-lists/description/

Solution:

Firstly, we choose our head, let's asume it's the first element in the first list. The first case is when the lists aren't empty (null), we the next node of the current node is the lower node between the two nodes.

The second case is when one list of them become empty or it's already empty from the beginning, so we add all the nodes of the other list directly to the current as the nodes in the lists are already sorted.

The third case is when the tow lists are empty (null) from the beginning, therefore the result is null.

```
class Solution {
  ListNode? mergeTwoLists(ListNode? list1, ListNode? list2) {
    ListNode? head = ListNode(0);
    ListNode? current = head;

    while (list1?.val != null && list2?.val != null) {
      if (list1!.val <= list2!.val) {
        current?.next = list1;
        list1 = list1.next;
      } else {
        current?.next = list2;
        list2 = list2.next;
      }

      current = current?.next;
    }

    if (list1 != null) {
      current?.next = list1;
    } else if (list2 != null) {
      current?.next = list2;
    }
    // Two lists are empty, then return null
    else {
      return null;
    }

    return head.next;
  }
}
```

Remove duplicates from sorted array

Problem Description:

https://leetcode.com/problems/remove-duplicates-from-sorted-array/description/

Solution:

We use two pointers **leftPointer** (L) and **rightPointer** (R) and make them starts from the index **1**, because the index 0 is always unique as it's the first element.

Assume we have this array [2, 3, 3, 4], we check if the current integer isn't equal to the previous integer, so at the first cycle we found that the condition is true so the value of the L becomes the value of the R which is the first **3** because both of them starts from index 1.

So the index of R and L increase to become 2**.**

In the second cycle, we found that the condition isn't true because the value of R is equal to the one previous it, so the index of R increases to become 3 and the index of L still the same (2).

In the third cycle, we found that the condition is true, therefore the value of L which equal 3 at index 2 becomes the value of R which is 4, then the value of L increases.

Now if we sum the times of increasing (moving) L, we'll find it's 3.

```
class Solution {
  int removeDuplicates(List<int> nums) {
    int leftPointer = 1;

    for (int rightPointer = 1; rightPointer < nums.length;
rightPointer++) {
      if (nums[rightPointer] != nums[rightPointer - 1]) {
        nums[leftPointer] = nums[rightPointer];
        leftPointer += 1;
      }
    }

    return leftPointer;
  }
}
```

Remove Element

Problem Description:

https://leetcode.com/problems/remove-element/

Solution:

```
class Solution {
  int removeElement(List<int> nums, int val) {
    for (int i = nums.length - 1; i >= 0; i--) {
      if (nums[i] == val) nums.remove(nums[i]);
    }

    return nums.length;
  }
}
```

Find the index of the first occurrence in a String

Problem Description:

https://leetcode.com/problems/find-the-index-of-the-first-occurrence-in-a-string/description/

Solution:

```
class Solution {
  int strStr(String haystack, String needle) {
    if (!haystack.contains(needle)) return -1;

    return haystack.indexOf(needle);
  }
}
```

Search Insert Position

Problem Description:

https://leetcode.com/problems/search-insert-position/

Solution:

```
class Solution {
  int searchInsert(List<int> nums, int target) {
    if (nums.isEmpty) {
      return -1;
    }

    int low = 0;
    int high = nums.length - 1;

    while (low <= high) {
      int mid = (low + high) ~/ 2;

      if (target == nums[mid]) return mid;

      if (target > nums[mid]) {
        low = mid + 1;
      } else {
        high = mid - 1;
      }
    }

    return low;
  }
}
```

Length of The Last Word

Problem Description:

https://leetcode.com/problems/length-of-last-word/

Solution:

```
class Solution {
  int lengthOfLastWord(String s) {
    List<String> listString = s.trim().split(' ');

    if (listString.length == 1) return listString[0].length;
```

```
    return listString[listString.length - 1].length;
  }
}
```

Plus One

Problem Description:

https://leetcode.com/problems/plus-one/

Solution:

```
class Solution {
  List<int> plusOne(List<int> digits) {
    int n = digits.length;

    // Iterate from right to left
    for (int i = n - 1; i >= 0; i--) {
      // If the current digit is less than 9, we can simply
increment it and return the digits array
      if (digits[i] < 9) {
        digits[i]++;
        return digits;
      }
      // Otherwise, set the current digit to 0 and continue
iterating to the left
      digits[i] = 0;
    }

    // If we've reached this point, it means that all digits were
9's, so we need to add a new leading 1
    digits.insert(0, 1);
    return digits;
  }
}
```

Another Solution:

```
class Solution {
  List<int> plusOne(List<int> digits) {
    String elements = "";
    List<int> plusedDigits = [];
```

```
    digits.forEach((element) {
      elements += element.toString();
    });

    int plusedOne = int.parse(elements) + 1;

    for (int i = 0; i < plusedOne.toString().length; i++) {
      plusedDigits.add(int.parse(plusedOne.toString()[i]));
    }

    return plusedDigits;
  }
}
```

Add Binary

Problem Description:

https://leetcode.com/problems/add-binary/description/

Solution:

```
class Solution {
  String addBinary(String a, String b) {
    int carry = 0;
    String result = '';

    // Ensure that two binary numbers are the same length by
adding zeros
    // to the left side of the shorter number
    while (a.length < b.length) {
      a = '0' + a;
    }

    while (b.length < a.length) {
      b = '0' + b;
    }

    // Iterate from right to left as it's a summation process
    for (int i = a.length - 1; i >= 0; i--) {
      int digitA = int.parse(a[i]);
      int digitB = int.parse(b[i]);
```

```
      int sum = digitA + digitB + carry;

      if (sum == 0 || sum == 1) {
        // Add sum to left because it's a summation process
        result = sum.toString() + result;
        carry = 0;
      } else if (sum == 2) {
        result = '0' + result;
        carry = 1;
      } else {
        result = '1' + result;
        carry = 1;
      }
    }

    // If there is a carry left over, add it to the left of the
result
    if (carry == 1) {
      result = '1' + result;
    }

    return result;
  }
}
```

Sqrt(x)

Problem Description:

https://leetcode.com/problems/sqrtx/description/

Solution:

```
class Solution {
  int mySqrt(int x) {
    int low = 0;
    int high = x;

    while (low <= high) {
      int mid = (low + high) ~/
          2; // Use integer division to get the floor of the
midpoint
```

```
      int square = mid * mid;

      if (square == x) {
        return mid;
      } else if (square > x) {
        high = mid - 1;
      } else {
        low = mid + 1;
      }
    }

    return high; // When the loop terminates, low > high, so
return high as the floor of the integer square root
  }
}
```

Climbing Stairs

Problem Description:

https://leetcode.com/problems/climbing-stairs/description/

Solution:

```
class Solution {
  int climbStairs(int n) {
    // Create a List with length = n, and fill it with Zeros
    List<int> nWays = List<int>.filled(n + 1, 0);

    // if n = 0 or 1 the steps = 1
    nWays[0] = 1;
    nWays[1] = 1;

    for (int i = 2; i <= n; i++) {
      nWays[i] = nWays[i - 1] + nWays[i - 2];
    }

    return nWays[n];
  }
}
```