Blind 75 Coding Questions

Mustafa Muhammad

27th December 2021

Array

1. Two Sum
2. Best Time to Buy and Sell Stock
3. Contains Duplicate
4. Product of Array Except Self
5. Maximum Subarray
6. Maximum Product Subarray
7. Find Minimum in Rotated Sorted Array
8. Search in Rotated Sorted Array
9. 3 Sum
10. Container With Most Water

Binary

1. Sum of Two Integers
2. Number of 1 Bits
3. Counting Bits
4. Missing Number
5. Reverse Bits

Dynamic Programming

1. Climbing Stairs
2. Coin Change
3. Longest Increasing Subsequence
4. Longest Common Subsequence
5. Word Break Problem
6. Combination Sum
7. House Robber
8. House Robber II
9. Decode Ways
10. Unique Paths
11. Jump Game

Graph

1. Clone Graph
2. Course Schedule
3. Pacific Atlantic Water Flow
4. Number of Islands
5. Longest Consecutive Sequence
6. Alien Dictionary (Leetcode Premium)
7. Graph Valid Tree (Leetcode Premium)
8. Number of Connected Components in an Undirected Graph (Leetcode Premium)

Interval

1. Insert Interval
2. Merge Intervals
3. Non-overlapping Intervals
4. Meeting Rooms (Leetcode Premium)
5. Meeting Rooms II (Leetcode Premium)

Matrix

1. Set Matrix Zeroes
2. Spiral Matrix
3. Rotate Image
4. Word Search

String

1. Longest Substring Without Repeating Characters
2. Longest Repeating Character Replacement
3. Minimum Window Substring
4. Valid Anagram
5. Group Anagrams
6. Valid Parentheses
7. Valid Palindrome
8. Longest Palindromic Substring
9. Palindromic Substrings
10. Encode and Decode Strings (Leetcode Premium)

Heap

1. Merge K Sorted Lists
2. Top K Frequent Elements
3. Find Median from Data Stream

LinkedList

1. Reverse a Linked List
2. Detect Cycle in a Linked List
3. Merge Two Sorted Lists
4. Merge K Sorted Lists
5. Remove Nth Node From End Of List
6. Reorder List

Tree

1. Maximum Depth of Binary Tree
2. Same Tree
3. Invert/Flip Binary Tree
4. Binary Tree Maximum Path Sum
5. Binary Tree Level Order Traversal
6. Serialize and Deserialize Binary Tree
7. Subtree of Another Tree
8. Construct Binary Tree from Preorder and Inorder Traversal
9. Validate Binary Search Tree
10. Kth Smallest Element in a BST
11. Lowest Common Ancestor of BST
12. Implement Trie (Prefix Tree)
13. Add and Search Word
14. Word Search II

1 Two Sum

Return index of two elements in the array that add upto the target value.

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        hash_map = {}

        for idx, num in enumerate(nums):
            res = target - num
            if res not in hash_map:
                hash_map[num] = idx
            else:
                return idx, hash_map[res]

        return -1, -1
```

2 Best Time To Buy and Sell Stock

Return the maximum profit that can be made using the price values provided.

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        profit = 0
        buying_price = float('inf')
        for i in range(0, len(prices)):
            buying_price = min(prices[i], buying_price)
            profit = max(profit, prices[i] - buying_price)
        return profit
```

3 Contains Duplicate

Return boolean value if there are duplicate elements in the array.

```
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        dup = set()
        for i in range(len(nums)):
            if nums[i] not in dup:
                dup.add(nums[i])
            else:
                return True
        return False
```

4 Product of Array Except Self

Input: nums = [1,2,3,4] Output: [24,12,8,6]

Input: nums = [-1,1,0,-3,3] Output: [0,0,9,0,0]

```
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        res = [1] * len(nums)


        prefix = 1
        for i in range(0, len(nums)):
            res[i] *= prefix
            prefix *= nums[i]

        postfix = 1
        for j in range(len(nums)-1, -1, -1):
            res[j] *= postfix
            postfix *= nums[j]

        return res
```

5 Maximum Subarray

Trick to solving is if the current sum is less than zero we set the current sum to zero.

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        #Kadane's algorithm repeated work
        max_sum = nums[0]
        curr_sum = 0
        for i in range(0, len(nums)):
            if curr_sum < 0:
                curr_sum = 0
            curr_sum += nums[i]
            max_sum = max(curr_sum , max_sum)
        return max_sum
```

6 Maximum Product Subarray

Technique to solving is to maintain current minimum and current maximum and utilize both to get the maximum answer.

```
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        #if the max subarray is just one value
        result = max(nums)
        curr_min = 1
        curr_max = 1
        for n in nums:
            temp = curr_max
            curr_max = max(curr_max*n, curr_min*n, n)
            curr_min = min(curr_min*n, temp*n, n)
            result = max(result, curr_max)
        return result
```

7 Container with most water

Two pointer approach and move in the direction of greater height.

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        right = 0
        left = len(height)-1
        max_area = 0
        while right < left:
            min_height = min(height[right], height[left])
            max_area = max(max_area, min_height * (left-right))
            if height[right] < height[left]:
                right+=1
            else:
                left -= 1
        return max_area
```

8 Find minimum in rotated sorted array

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        # binary search
        left = 0
        right = len(nums)-1
        while left <= right:
            mid = int(left + (right-left)/2)

            next = (mid+1)%len(nums)
            prev = (mid-1+len(nums))%len(nums)

            if nums[mid] <= nums[prev] and nums[mid] <= nums[next]:
                return nums[mid]
            elif nums[0] <= nums[mid]:
                left = mid + 1
            elif nums[mid] <= nums[-1]:
                right = mid -1
        return nums[0]
```

9 Search in rotated sorted array

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:

        def find_min_index(nums):
            left = 0
            right = len(nums)-1

            while left <= right:
                mid = int(left + (right-left)/2)

                next = (mid+1)%len(nums)
                prev = (mid-1+len(nums))%len(nums)

                if nums[mid] <= nums[prev] and nums[mid] <= nums[next]:
                    return mid

                elif nums[0] <= nums[mid]:
                    left = mid + 1
                elif nums[mid] <= nums[-1]:
                    right = mid -1

            return -1

        def binary_search(nums, target):
            left = 0
            right = len(nums)-1

            while left <= right:
                mid = int(left + (right-left)/2)

                if nums[mid] == target:
                    return mid

                elif nums[mid] < target:
                    left = mid + 1

                elif nums[mid] > target:
                    right = mid -1
            return -1

        ind = find_min_index(nums)

        arr1 = nums[0:ind]
        arr2 = nums[ind:]

        val = binary_search(arr1, target)
        val2 = binary_search(arr2, target)

        return val if val2 == -1 else val2+len(arr1)
```

10 3 Sum

Sort the array and run two sum II for each unique element in the loop.

```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        res = []
        nums = sorted(nums)
        for i in range(0, len(nums)):
            if i > 0 and nums[i] == nums[i-1]:
                continue
            left = i+1
            right = len(nums)-1

            while left < right:
                summed = nums[left] + nums[right] + nums[i]
                if summed == 0:
                    res.append((nums[left], nums[right], nums[i]))
                    left += 1
                    while nums[left] == nums[left -1] and left < right:
                        left += 1

                elif summed < 0:
                    left += 1

                elif summed > 0:
                    right -=1
        return res
```

11 Container with most water

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        left = 0
        right = len(height)-1
        max_area = -1
        while left < right:
            h = min(height[left], height[right])
            area = (right - left) * h
            max_area = max(max_area, area)
            if height[left] < height[right]:
                left += 1
            elif height[right] < height[left]:
                right -=1
            elif height[right] == height[left]:
                left+=1
        return max_area
```

12 Binary - Sum of two integers

We make use of xor a=1, b=0, ans=1. a=1, b=1, ans=0

When we have two one's we have a carry. If a and b are 1, we have a carry. a&b left shift 1

If we dont have a carry we are done.

Code doesnt work in python since the assumption that we make is that the length of integers is 32 bits. However python allows for an infinite length of integers. Java does not.

```
class Solution {
    public int getSum(int a, int b) {
        while( b!= 0){
            int temp = (a &b) <<1;
            a = a ^ b;
            b = temp;
        }
        return a;
    }
}
```

13 Number of one bits

Count the number of 1's in bits.

```
class Solution:
    def hammingWeight(self, n: int) -> int:
        total = 0
        while n != 0:
            total += n %2
            n = n >> 1
        return total
```

14 Counting Bits

Dynamic Programming Solution

```
class Solution:
    def countBits(self, n: int) -> List[int]:
        dp = [0]*(n+1)
        offset = 1

        for i in range(1, n+1):
            if offset * 2 == i:
                offset = i

            dp[i] = 1 + dp[i - offset]

        return dp
```

15 Missing Number

```
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        xor1 = 0
        xor2 = nums[0]
        for i in range(1, len(nums)+1):
            xor1 ^= i
        for j in range(1, len(nums)):
            xor2 ^= nums[j]

        return xor1 ^ xor2
```

16 Reverse Bits

```
class Solution:
    def reverseBits(self, n: int) -> int:
        res = 0
        for i in range(0, 32):
            res = res << 1
            if n % 2 == 1:
                res += 1

            n = n >> 1
        return res
```

17 DP - Climbing stairs

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

```
class Solution:
    def __init__(self):
        self.map = {}

    def climbStairs(self, n: int) -> int:
        if n == 0:
            return 0
        if n == 1:
            return 1
        if n == 2:
            return 2

        if n in self.map:
            return self.map[n]

        self.map[n] = self.climbStairs(n-1) + self.climbStairs(n-2)

        return self.map[n]
```

18 Coin Change

```
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        memo = {}

        def unbounded(coins, i, target):
            key = (i, target)

            if key in memo:
                return memo[key]

            if target == 0:
                return 0

            if i<=0 and target > 0:
                return float('inf')

            if coins[i-1] > target:
                memo[key] = unbounded(coins, i-1, target)
                return memo[key]

            memo[key] = min(1+unbounded(coins, i, target-coins[i-1]),
                            unbounded(coins, i-1, target))
            return memo[key]

        res = unbounded(coins, len(coins), amount)

        if res == float('inf'):
            return -1

        return res
```

19 Longest Increasing Subsequence

Recursive memoized solution.

Best to explore using nested loops as well.

```
class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        memo = {}

        def lcs(nums, prev, current):
            key = (prev, current)

            if key in memo:
                return memo[key]

            if len(nums) == current:
                return 0

            c1 = 0

            if (prev == -1) or (nums[prev] < nums[current]):
                c1 = 1+lcs(nums, current, current+1)

            c2 = lcs(nums, prev, current+1)

            memo[key] = max(c1, c2)

            return memo[key]

        return lcs(nums, -1, 0)
```

20 Longest Common Subsequence

```
class Solution:
    def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        memo = {}

        def lcs(s1, s2, i, j):
            key = (i, j)

            if key in memo:
                return memo[key]

            if i<=0 or j<=0:
                return 0

            if s1[i-1] == s2[j-1]:
                memo[key] = 1+lcs(s1, s2, i-1, j-1)
                return memo[key]

            memo[key] = max(lcs(s1, s2, i, j-1), lcs(s1, s2, i-1, j))

            return memo[key]

        return lcs(text1, text2, len(text1), len(text2))
```

21 Word Break

```
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        dp = [False] * (len(s)+1)
        dp[len(s)] = True

        for i in range(len(s), -1, -1):
            for w in wordDict:
                if i+len(w) <= len(s) and s[i : i + len(w)] == w:
                    dp[i] = dp[i+len(w)]

                if dp[i]:
                    break

        return dp[0]
```

Recursive DP

```
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        memo = {}

        def wb(s, wordDict):
            if s in memo:
                return memo[s]
            if len(s) == 0:
                memo[””] = True
                return True

            for word in wordDict:
                if s.startswith(word) and len(s) >= len(word):
                    if wb(s[len(word):], wordDict):
                        return True

            memo[s] = False

            return False

        return wb(s, wordDict)
```

22 Combination Sum

Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order.

Time Complexity = $2\hat{T}$ (Where T is the target value, as the height of the decision tree is determined by the target value.)

```
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        res = []

        def dfs(i, curr, total):
            if total == target:
                res.append(curr.copy())
                return

            if i>= len(candidates) or total > target:
                return

            curr.append(candidates[i])
            dfs(i, curr, total+candidates[i])
            curr.pop()
            dfs(i+1, curr, total)

            return

        dfs(0, [], 0)
        return res
```

23 House Robbers

Find the recurrence, code up the recursion.

Recurrence = max(arr[0] + dp(arr[2:]), dp(arr[1:]))

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        # recurrence
        #rob = max(nums[0] + rob(nums[2:]), rob(nums[1:]))
        memo  = {}

        def dp(nums, n):
            if n >= len(nums):
                return 0

            if n in memo:
                return memo[n]

            memo[n] = max(nums[n] + dp(nums, n+2), dp(nums, n+1))

            return memo[n]

        return dp(nums, 0)
```

24 House Robber II

This time the first and last house are connected so we split the array into two parts.

We run our house robber I function on arr[:-1] and arr[1:]

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        if len(nums) == 1:
            return nums[0]

        memo = {}

        def dp(nums, n):
            if n >= len(nums):
                return 0
            if n in memo:
                return memo[n]

            memo[n] = max(nums[n] + dp(nums, n+2), dp(nums, n+1))

            return memo[n]
```

```
arr1 = dp(nums[:-1], 0)
memo = {}
arr2 =  dp(nums[1:], 0)
max_ret = max(arr1, arr2)
return max_ret
```