[53. Maximum Subarray](https://leetcode.com/problems/maximum-subarray/)

[**labuladong 题解**](https://labuladong.github.io/article/slug.html?slug=maximum-subarray)[**思路**](https://leetcode.com/problems/maximum-subarray/)

Medium

32.2K

1.3K

Companies

Given an integer array nums, find the

subarray

 with the largest sum, and return *its sum*.

**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.

**Example 2:**

**Input:** nums = [1]

**Output:** 1

**Explanation:** The subarray [1] has the largest sum 1.

**Example 3:**

**Input:** nums = [5,4,-1,7,8]

**Output:** 23

**Explanation:** The subarray [5,4,-1,7,8] has the largest sum 23.

class Solution:

    def maxSubArray(self, nums: List[int]) -> int:

        left, right = 0, 0

        windowSum, maxSum = 0, float('-inf')

        while right < len(nums):

            windowSum += nums[right]

            right += 1

            maxSum = max(windowSum, maxSum)

            while windowSum < 0:

                windowSum -= nums[left]

                left += 1

        return maxSum

[198. House Robber](https://leetcode.com/problems/house-robber/)

Medium

19.4K

360

Companies

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array nums representing the amount of money of each house, return *the maximum amount of money you can rob tonight* ***without alerting the police***.

**Example 1:**

**Input:** nums = [1,2,3,1]

**Output:** 4

**Explanation:** Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

**Example 2:**

**Input:** nums = [2,7,9,3,1]

**Output:** 12

**Explanation:** Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob = 2 + 9 + 1 = 12.

class Solution:

    def rob(self, nums: List[int]) -> int:

        if len(nums) == 0: return 0

        if len(nums) == 1: return nums[0]

        if len(nums) == 2: return max(nums)

        # dynamic programming - decide each problem by its sub-problems:

        dp = [0]\*len(nums)

        dp[0] = nums[0]

        dp[1] = max(nums[0], nums[1])

        for i in range(2, len(nums)):

            dp[i] = max(dp[i-1], nums[i]+dp[i-2])

        return dp[-1]

1. write an algo to check if a string is sorted in alphabetical order and print 0 if it is, if not print the index of the character where it is out of the alphabetical order

def check\_alphabetical\_order(s):

for i in range(1, len(s)):

if s[i] < s[i - 1]:

return i

return 0

2. write an algo to find the sub-string from the given string that is the same when read forwards and backwards. Input: The input consists of a string-inputStr, from the given string, print a sub-string which is the same when read forwards and backwards. If there are multiple substrings of equal length, choose the lexicographically smallest one. If they are in different length, choose the one with maximum length. If there is no such substring, return none. Substring is only valid one its length is more than 1

def find\_longest\_lexicographically\_smallest\_palindrome(inputStr):

def is\_palindrome(s):

return s == s[::-1]

longest\_palindrome = ""

for i in range(len(inputStr)):

for j in range(i + 1, len(inputStr) + 1):

substring = inputStr[i:j]

if is\_palindrome(substring):

if len(substring) > len(longest\_palindrome) or (len(substring) == len(longest\_palindrome) and substring < longest\_palindrome):

longest\_palindrome = substring

if len(longest\_palindrome) > 1:

return longest\_palindrome

else:

return None

3. Lucy loves to play the Hop, Skip, and Jump game. Given an N\*M matrix and starting from cell(1, 1), her challenge is to hop in an anti-clockwise direction and skip alternate cells. The goal is to find out the last cell she would hop onto. Write the algo to find the last cell she would hop onto after moving anti-clockwise and skipping alternate cells. The inputs consist 2 integers - matrix\_row and matrix\_col, representing the number of rows(N) and the number of columns (M) in the matrix. The next M lines consist of N space-separated integers representing the elements in each cell of the matrix. Output: print an integer representing the last cell Lucy would hop onto

def funcHopSkipJump(matrix):

n = len(matrix)

m = len(matrix[0])

total\_cells = n \* m

skip\_count = (total\_cells + 1) // 2 # Calculate how many cells to skip

# Calculate the position using the skip\_count

x = skip\_count // m

y = skip\_count % m

return matrix[x][y]

# Main function

def main():

# Input for matrix

matrix = []

matrix\_rows, matrix\_cols = map(int, input().split())

for idx in range(matrix\_rows):

matrix.append(list(map(int, input().split())))

result = funcHopSkipJump(matrix)

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

main()

below exceeds time limit

matrix\_rows = len(matrix)

    matrix\_cols = len(matrix[0])

    def is\_valid(row, col):

        return 0 <= row < matrix\_rows and 0 <= col < matrix\_cols

    dx = [0, 1, 0, -1]

    dy = [1, 0, -1, 0]

    direction = 0

    row, col = 0, 0

    while True:

        new\_row = row + dx[direction]

        new\_col = col + dy[direction]

        if is\_valid(new\_row, new\_col):

             row, col = new\_row, new\_col

        else:

            break

        direction = (direction + 1) % 4

    return matrix[row][col]