180208176209135208184-208161-zii

December 23, 2023

[ ]:
```
good evening. hope its all good. i tried my best.
```

— , .
. , .

— .

root , .

1:

: root = [1,2,3] : 6 : — 2 -> 1 -> 3 2 + 1 + 3 = 6.

2:

: root = [-10,9,20,null,null,15,7] : 42 : — 15 -> 20 -> 7 15 + 20 + 7 = 42.

:

.[1, 3 * 104] -1000 <= Node.val <= 1000

[ ]:
```
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

def maxPathSum(root):
    def maxSum(node):
        nonlocal max_sum
        if not node:
            return 0
        left_sum = max(maxSum(node.left), 0)
        right_sum = max(maxSum(node.right), 0)
        max_sum = max(max_sum, left_sum + right_sum + node.value)
        return max(left_sum, right_sum) + node.value

    max_sum = float('-inf')
    maxSum(root)
    return max_sum
```

```python
root1 = TreeNode(1, TreeNode(2), TreeNode(3))
root2 = TreeNode(-10, TreeNode(9), TreeNode(20, TreeNode(15), TreeNode(7)))

print(maxPathSum(root1))
print(maxPathSum(root2))
```

```
6
42
```

2

nums indexDiff valueDiff.

(i, j) , :

i != j, abs(i - j) <= indexDiff. abs(nums[i] - nums[j]) <= valueDiff, , true false .

1:

:

```
 nums = [1,2,3,1], indexDiff = 3, valueDiff = 0
    : true
       :            (i, j) = (0, 3).
```

:

```
i != j --> 0 != 3

abs(i - j) <= indexDiff --> abs(0 - 3) <= 3
abs(nums[i] - nums[j]) <= valueDiff --> abs(1 - 1) <= 0
```

2:

```
: nums = [1,5,9,1,5,9], indexDiff = 2, valueDiff = 3

  : false
```

: (i, j), , .

:

```
2 <= nums.length <= 10^5

-10^9 <= nums[i] <= 10^9

1 <= indexDiff <= nums.length

0 <= valueDiff <= 10^9
```

[ ]:

```python
def containsNearbyAlmostDuplicate(nums, indexDiff, valueDiff):
    if len(nums) <= 1:
        return False
    for i in range(len(nums)):
        for j in range(i+1, min(i+indexDiff+1, len(nums))):
            if abs(nums[i] - nums[j]) <= valueDiff:
                return True
    return False

nums1 = [1, 2, 3, 1]
indexDiff1 = 3
valueDiff1 = 0
print(containsNearbyAlmostDuplicate(nums1, indexDiff1, valueDiff1))

nums2 = [1, 5, 9, 1, 5, 9]
indexDiff2 = 2
valueDiff2 = 3
print(containsNearbyAlmostDuplicate(nums2, indexDiff2, valueDiff2))
```

```
True
False
```

3

— . , , —
.

, arr = [2,3,4] 3.

, arr = [2,3] (2 + 3) / 2 = 2.5.

MedianFinder:

MedianFinder() MedianFinder .

void addNum(int num) num .

double findMedian() . .10-5

1:

```
["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]
[[], [1], [2], [], [3], []]

[  ,   ,   , 1,5,   , 2,0]
```

```
MedianFinder medianFinder =     MedianFinder();
medianFinder.addNum(1); //    = [1]
medianFinder.addNum(2); //    = [1, 2]
medianFinder.findMedian(); //       1,5 ( . . (1 + 2)/2)
medianFinder.addNum(3); //   [1, 2, 3]
medianFinder.findMedian(); //        2.0
```

:

```
-105 <= num <= 105
```

findMedian. “‘ .5

* 10ˆ4 addNum findMedian

“‘ :

[0, 100], ? 99%
[0, 100], ?

```python
import heapq

class MedianFinder:

    def __init__(self):
        self.max_heap = []
        self.min_heap = []

    def addNum(self, num: int) -> None:
        if not self.max_heap or num <= -self.max_heap[0]:
            heapq.heappush(self.max_heap, -num)
        else:
            heapq.heappush(self.min_heap, num)

        if len(self.max_heap) > len(self.min_heap) + 1:
            heapq.heappush(self.min_heap, -heapq.heappop(self.max_heap))
        elif len(self.min_heap) > len(self.max_heap):
            heapq.heappush(self.max_heap, -heapq.heappop(self.min_heap))

    def findMedian(self) -> float:
        if len(self.max_heap) == len(self.min_heap):
            return (-self.max_heap[0] + self.min_heap[0]) / 2
        else:
            return -self.max_heap[0]
```

```python
#             MedianFinder,                                              Python.␣
↪    addNum                    ,       findMedian                              .   ␣
↪                  ,                                         0    100,         ␣
↪                                   ,                .      ,     99%␣
↪                  0    100,                                                    ␣
↪                                      .
```

4 n — , n n x n ,
.

n, n . .

n , ‘Q’ ’.’ .

1:

: n = 4

: [[".Q..","...Q","Q...","..Q."],["..Q.","Q.. .","...Q",".Q.."]]
: , .

2:

: n = 1
: [["Q"]]

:

1 <= n <= 9

[ ]:

```
def solveNQueens(n):
    def could_place(row, col):
        return not (cols[col] + left_diag[row - col] + right_diag[row + col])

    def place_queen(row, col):
        queens_pos.append((row, col))
        cols[col] = 1
        left_diag[row - col] = 1
        right_diag[row + col] = 1

    def remove_queen(row, col):
        queens_pos.pop()
        cols[col] = 0
        left_diag[row - col] = 0
        right_diag[row + col] = 0

    def add_solution():
        sol = []
        for _, col in queens_pos:
            sol.append('.'*col + 'Q' + '.'*(n - col - 1))
        solutions.append(sol)

    def backtrack(row):
        for col in range(n):
            if could_place(row, col):
                place_queen(row, col)
                if row + 1 == n:
                    add_solution()
                else:
                    backtrack(row + 1)
                remove_queen(row, col)

    cols = [0] * n
    left_diag = [0] * (2 * n - 1)
    right_diag = [0] * (2 * n - 1)
    queens_pos = []
    solutions = []

    backtrack(0)
    return solutions

print(solveNQueens(4))
print(solveNQueens(1))
```

```
[['.Q..', '…Q', 'Q…', '..Q.'], ['..Q.', 'Q…', '…Q', '.Q..']]
[['Q']]
```

5

5: rows x cols matrix, 0’s 1’s, ,
1’s, .

1:

:

= [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0'

:

6. : .

2:

: = [["0"]]
: 0
3:

: = [["1"]]
: 1

:

rows == matrix.length
cols == matrix[i].length
1 <= row, cols <= 200
matrix[i][j] '0' '1'.

[ ]:
```
def maximalRectangle(matrix):
    if not matrix:
        return 0

    heights = [0] * len(matrix[0])
    max_area = 0

    def max_area_in_histogram(heights):
        stack = [-1]
        area = 0
        for i in range(len(heights)):
            while stack[-1] != -1 and heights[i] <= heights[stack[-1]]:
                h = heights[stack.pop()]
                w = i - stack[-1] - 1
                area = max(area, h * w)
            stack.append(i)
        while stack[-1] != -1:
            h = heights[stack.pop()]
            w = len(heights) - stack[-1] - 1
            area = max(area, h * w)
        return area

    for row in matrix:
        for i, val in enumerate(row):
            heights[i] = heights[i] + 1 if val == "1" else 0
        max_area = max(max_area, max_area_in_histogram(heights))
    return max_area

matrix1 =␣
 ↪[["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]
print(maximalRectangle(matrix1))

matrix2 = [["0"]]
print(maximalRectangle(matrix2))

matrix3 = [["1"]]
print(maximalRectangle(matrix3))
```

```
6
0
1
```

6

prices, prices[i] .ith

, . .

. (. . ,).

1:

```
   :    = [3,3,5,0,0,3,1,4]
     : 6
:             4 (   = 0)                6 (   = 3),      = 3- 0 = 3.
           7 (   = 1)             8 (   = 4),      = 4-1 = 3.
```

2:

: = [1,2,3,4,5]
: 4
: 1 (= 1) 5 (= 5), = 5-1 = 4.
, , ,

3:

: = [7,6,4,3,1]
: 0
: , . . = 0.

:

1 <= prices.length <= 105
0 <= prices[i] <= 105

[ ]:
```python
def maxProfit(prices):
    if not prices:
        return 0

    n = len(prices)
    max_profit_one = [0] * n
    max_profit_two = [0] * n

    min_price = prices[0]
    for i in range(1, n):
        min_price = min(min_price, prices[i])
        max_profit_one[i] = max(max_profit_one[i-1], prices[i] - min_price)

    max_price = prices[n-1]
    for i in range(n-2, -1, -1):
        max_price = max(max_price, prices[i])
        max_profit_two[i] = max(max_profit_two[i+1], max_price - prices[i])

    max_profit = max_profit_two[0]
    for i in range(1, n):
        max_profit = max(max_profit, max_profit_one[i-1] + max_profit_two[i])

    return max_profit

prices1 = [3, 3, 5, 0, 0, 3, 1, 4]
print(maxProfit(prices1))

prices2 = [1, 2, 3, 4, 5]
print(maxProfit(prices2))

prices3 = [7, 6, 4, 3, 1]
print(maxProfit(prices3))
```

```
6
```

```
4
0
```

[ ]:

[ ]: