[61. Rotate List](https://leetcode.com/problems/rotate-list/)

class Solution:

def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:

if not head or k==0:

return head

tail = head

print(tail.val)

length = 1

while tail.next:

length+=1

tail = tail.next

k = k%length

if k==0:

return head

dummy = ListNode(0)

dummy.next = head

current = dummy

for \_ in range(length-k):

current = current.next

newHead = current.next

current.next = None

tail.next = head

return newHead

[62. Unique Paths](https://leetcode.com/problems/unique-paths/)

class Solution:

def uniquePaths(self, m: int, n: int) -> int:

dp = [[0]\*n for \_ in range(m)]

for i in range(n):

dp[m-1][i]=1

for i in range(m):

dp[i][n-1]=1

for i in range(m-2,-1,-1):

for j in range(n-2,-1,-1):

dp[i][j] = dp[i+1][j]+dp[i][j+1]

return dp[0][0]

[63. Unique Paths II](https://leetcode.com/problems/unique-paths-ii/)

class Solution:

def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:

m = len(obstacleGrid)

n = len(obstacleGrid[0])

dp = [[0]\* (n+1) for \_ in range(m+1)]

dp[0][1]=1

for i in range(1,m+1):

for j in range(1, n+1):

if not obstacleGrid[i-1][j-1]:

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

return dp[-1][-1]

[64. Minimum Path Sum](https://leetcode.com/problems/minimum-path-sum/)

class Solution:

def minPathSum(self, grid: List[List[int]]) -> int:

m = len(grid)

n = len(grid[0])

dp = [[float("inf")]\* (n+1) for \_ in range(m+1)]

dp[m-1][n]=0

for i in range(m-1,-1,-1):

for j in range(n-1,-1,-1):

dp[i][j]= grid[i][j]+min(dp[i][j+1], dp[i+1][j])

return dp[0][0]

[65. Valid Number](https://leetcode.com/problems/valid-number/)

class Solution:

def isNumber(self, s: str) -> bool:

try:

if 'inf' in s.lower() or s.isalpha():

return False

if float(s) or float(s)>=0:

return True

except ValueError:

return False

[66. Plus One](https://leetcode.com/problems/plus-one/)

class Solution:

def plusOne(self, digits: List[int]) -> List[int]:

carry = 1

for i in range(len(digits)-1,-1,-1):

total = digits[i]+carry

carry = total//10

digits[i] = total%10

if total<10:

return digits

if carry:

digits.insert(0, carry)

return digits

[67. Add Binary](https://leetcode.com/problems/add-binary/)

class Solution:

def addBinary(self, a: str, b: str) -> str:

i = len(a)-1

j = len(b)-1

carry = 0

res = ''

while i>=0 or j>=0:

aBit = int(a[i]) if i>=0 else 0

bBit = int(b[j]) if j>=0 else 0

addition = aBit + bBit + carry

res = str(addition%2)+res

carry = addition//2

i=i-1

j=j-1

if carry:

res = str(carry)+res

return res

[69. Sqrt(x)](https://leetcode.com/problems/sqrtx/)

class Solution:

def mySqrt(self, x: int) -> int:

if x<2:

return x

l = 1

r = x//2

while l<=r:

mid = (l+r)//2

if mid \* mid == x:

return mid

elif mid\*mid <x:

l = mid+1

else:

r = mid -1

return r

[367. Valid Perfect Square](https://leetcode.com/problems/valid-perfect-square/)

class Solution:

def isPerfectSquare(self, num: int) -> bool:

if num<2:

return True

l= 1

r = num//2

while l<=r:

mid = (l+r)//2

if mid\*mid ==num:

return True

elif mid\*mid<num:

l = mid+1

else:

r = mid-1

return False

[70. Climbing Stairs](https://leetcode.com/problems/climbing-stairs/)

class Solution:

def climbStairs(self, n: int) -> int:

if n==0 or n==1 or n==2:

return n

dp = [0]\*(n+1)

dp[1]=1

dp[2]=2

print(dp)

for i in range(3, n+1):

dp[i] = dp[i-1]+dp[i-2]

return dp[-1]

[509. Fibonacci Number](https://leetcode.com/problems/fibonacci-number/)

class Solution:

def fib(self, n: int) -> int:

if n==0:

return 0

if n==1 or n==2:

return 1

dp = [0]\*(n+1)

dp[1]=1

dp[2]=1

for i in range(3,n+1):

dp[i]=dp[i-1]+dp[i-2]

return dp[-1]

[1137. N-th Tribonacci Number](https://leetcode.com/problems/n-th-tribonacci-number/)

class Solution:

def tribonacci(self, n: int) -> int:

if n==0:

return 0

if n==1 or n==2:

return 1

dp = [0]\*(n+1)

dp[1]=1

dp[2]=1

for i in range(3,n+1):

dp[i]=dp[i-1]+dp[i-2]+dp[i-3]

return dp[-1]

[71. Simplify Path](https://leetcode.com/problems/simplify-path/)

class Solution:

def simplifyPath(self, path: str) -> str:

# components = path.split("/")

stack = []

for component in path.split("/"):

if component=="..":

if stack:

stack.pop()

elif component and component!='.':

stack.append(component)

simplifiedPath = '/'+'/'.join(stack)

return simplifiedPath

[72. Edit Distance](https://leetcode.com/problems/edit-distance/)

|  | word1 | i | n | t | e | n | t | i | o | n |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| word2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| e | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| x | 2 | 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| e | 3 | 3 | 3 | 3 | 3 | 4 | 5 | 6 | 7 | 8 |
| c | 4 | 4 | 4 | 4 | 4 | 4 | 5 | 6 | 7 | 8 |
| u | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 6 | 7 | 8 |
| t | 6 | 6 | 6 | 5 | 6 | 6 | 5 | 6 | 7 | 8 |
| i | 7 | 6 | 7 | 6 | 6 | 7 | 6 | 5 | 6 | 7 |
| o | 8 | 7 | 7 | 7 | 7 | 7 | 7 | 6 | 5 | 6 |
| n | 9 | 8 | 8 | 8 | 8 | 7 | 8 | 7 | 6 | 5 |

class Solution:

def minDistance(self, word1: str, word2: str) -> int:

m = len(word1)

n= len(word2)

dp= [[0]\*(m+1) for \_ in range(n+1)]

for i in range(m+1):

dp[0][i]=i

for j in range(n+1):

dp[j][0]=j

print(dp)

for r in range(1,n+1):

for c in range(1,m+1):

if word1[c-1]==word2[r-1]:

dp[r][c]=dp[r-1][c-1]

else:

dp[r][c]= 1+min(dp[r-1][c], dp[r][c-1], dp[r-1][c-1])

return dp[-1][-1]

[73. Set Matrix Zeroes](https://leetcode.com/problems/set-matrix-zeroes/)

class Solution:

def setZeroes(self, matrix: List[List[int]]) -> None:

"""

Do not return anything, modify matrix in-place instead.

"""

m = len(matrix)

n = len(matrix[0])

rows=set()

cols =set()

for i in range(m):

for j in range(n):

if matrix[i][j]==0:

rows.add(i)

cols.add(j)

for i in range(m):

for j in range(n):

if i in rows or j in cols:

matrix[i][j]=0

[74. Search a 2D Matrix](https://leetcode.com/problems/search-a-2d-matrix/)

class Solution:

def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:

rows = len(matrix)

cols = len(matrix[0])

row = 0

col = cols-1

while row<rows and col>=0:

if matrix[row][col]==target:

return True

elif matrix[row][col]<target:

row = row+1

else:

col = col - 1

return False

[240. Search a 2D Matrix II](https://leetcode.com/problems/search-a-2d-matrix-ii/)

class Solution:

def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:

r = 0

c= len(matrix[0])-1

while r<len(matrix) and c>-1:

if matrix[r][c]==target:

return True

elif matrix[r][c]<target:

r= r+1

else:

c = c-1

return False

[75. Sort Colors](https://leetcode.com/problems/sort-colors/)

class Solution:

def sortColors(self, nums: List[int]) -> None:

"""

Do not return anything, modify nums in-place instead.

"""

l = 0

r = len(nums)-1

current = 0

while current<=r:

if nums[current]==0:

nums[current], nums[l]=nums[l], nums[current]

l+=1

current+=1

elif nums[current]==2:

nums[r],nums[current]=nums[current],nums[r]

r-=1

else:

current+=1

[76. Minimum Window Substring](https://leetcode.com/problems/minimum-window-substring/)

class Solution:

def minWindow(self, s: str, t: str) -> str:

from collections import Counter

if not s or not t:

return ""

l = 0

r = 0

targetDic = Counter(t)

windowDic = {}

requiredChars = len(targetDic)

formedChars = 0

minWindow = ''

minLen = float('inf')

while r<len(s):

char = s[r]

windowDic[char] = windowDic.get(char,0)+1

if char in targetDic and targetDic[char]==windowDic[char]:

formedChars+=1

while formedChars==requiredChars:

if r-l+1<minLen:

minLen = r-l+1

minWindow=s[l:r+1]

leftChar = s[l]

windowDic[leftChar]-=1

if leftChar in targetDic and windowDic[leftChar]<targetDic[leftChar]:

formedChars-=1

l+=1

r+=1

return minWindow

[77. Combinations](https://leetcode.com/problems/combinations/)

class Solution:

def combine(self, n: int, k: int) -> List[List[int]]:

res = []

def backtrack(nums,path):

if len(path)==k:

res.append(path)

return

for i in range(len(nums)):

backtrack(nums[i+1:], path+[nums[i]])

backtrack(list(range(1,n+1)), [])

return res

[78. Subsets](https://leetcode.com/problems/subsets/)

class Solution:

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

res = []

def backtrack(nums,path):

res.append(path)

for i in range(len(nums)):

backtrack(nums[i+1:], path+[nums[i]])

backtrack(nums,[])

return res

[79. Word Search](https://leetcode.com/problems/word-search/)

class Solution:

def exist(self, board: List[List[str]], word: str) -> bool:

row = len(board)

col = len(board[0])

path = set()

def dfs(r,c,i):

if i==len(word):

return True

if (r<0 or c<0 or r>=row or c>=col or word[i]!=board[r][c] or (r,c)in path):

return False

path.add((r,c))

res = (dfs(r+1,c,i+1)or

dfs(r-1, c,i+1) or

dfs(r,c+1,i+1) or

dfs(r,c-1,i+1)

)

path.remove((r,c))

return res

for r in range(row):

for c in range(col):

if dfs(r,c,0):

return True

return False

[80. Remove Duplicates from Sorted Array II](https://leetcode.com/problems/remove-duplicates-from-sorted-array-ii/)

class Solution:

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

k = 2

for i in range(2, len(nums)):

if nums[i]!=nums[k-2]:

nums[k]=nums[i]

k=k+1

return k