1.valid sudoku

#include <stdbool.h>

bool isValidSudoku(char\*\* board, int boardSize, int\* boardColSize) {

int rows[9][9] = {0};

int cols[9][9] = {0};

int boxes[3][3][9] = {0};

for (int i = 0; i < boardSize; i++) {

for (int j = 0; j < \*boardColSize; j++) {

if (board[i][j] != '.') {

int num = board[i][j] - '0' - 1;

if (rows[i][num]++ || cols[j][num]++ || boxes[i / 3][j / 3][num]++) {

return false;

}

}

}

}

return true;

}

2.sudoku solver

class Solution:

def solveSudoku(self, board):

def is\_valid(num, row, col):

for i in range(9):

if board[i][col] == num or board[row][i] == num or board[3 \* (row // 3) + i // 3][3 \* (col // 3) + i % 3] == num:

return False

return True

def solve():

for i in range(9):

for j in range(9):

if board[i][j] == '.':

for num in '123456789':

if is\_valid(num, i, j):

board[i][j] = num

if solve():

return True

board[i][j] = '.'

return False

return True

solve()

3.n queens

def is\_safe(board, row, col, N):

for i in range(col):

if board[row][i] == 1:

return False

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

return False

for i, j in zip(range(row, N, 1), range(col, -1, -1)):

if board[i][j] == 1:

return False

return True

def solve\_n\_queens\_util(board, col, N):

if col >= N:

return True

for i in range(N):

if is\_safe(board, i, col, N):

board[i][col] = 1

if solve\_n\_queens\_util(board, col + 1, N) == True:

return True

board[i][col] = 0

return False

def solve\_n\_queens(N):

board = [[0 for \_ in range(N)] for \_ in range(N)]

if solve\_n\_queens\_util(board, 0, N) == False:

return False

return board

def print\_solution(board):

for row in board:

print(row)

# Example usage

N = 4

solution = solve\_n\_queens(N)

if solution:

print\_solution(solution)

else:

print("No solution exists for N = ", N)

4.n queens ||

class Solution:

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

def backtrack(queens, xy\_diff, xy\_sum):

p = len(queens)

if p == n:

output.append(queens)

return

for q in range(n):

if q not in queens and p - q not in xy\_diff and p + q not in xy\_sum:

backtrack(queens + [q], xy\_diff + [p - q], xy\_sum + [p + q])

output = []

backtrack([], [], [])

return len(output)

5.subsets

from itertools import combinations

def subsets(nums):

res = []

for i in range(len(nums)+1):

res.extend(list(combinations(nums, i)))

return [list(subset) for subset in res]

# Example 1

nums1 = [1, 2, 3]

print(subsets(nums1))

# Example 2

nums2 = [0]

print(subsets(nums2))

6.longest palindromic string

class Solution:

def longestPalindrome(self, s: str) -> str:

def expandAroundCenter(left, right):

while left >= 0 and right < len(s) and s[left] == s[right]:

left -= 1

right += 1

return s[left + 1:right]

if len(s) < 1:

return ""

longest = ""

for i in range(len(s)):

palindrome1 = expandAroundCenter(i, i)

palindrome2 = expandAroundCenter(i, i + 1)

longest = max(longest, palindrome1, palindrome2, key=len)

return longest