1. def solve\_n\_queens(N):

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

return all(board[i][col] == 0 and (board[i][j] == 0 for j in range(N) if abs(row - i) == abs(col - j)) for i in range(row))

def solve\_n\_queens\_util(board, row):

if row == N:

for row in board:

print("".join("Q" if x else "." for x in row))

print("\n")

return

for col in range(N):

if is\_safe(board, row, col):

board[row][col] = 1

solve\_n\_queens\_util(board, row + 1)

board[row][col] = 0

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

solve\_n\_queens\_util(board, 0)

print("Solutions for N=4:")

solve\_n\_queens(4)

print("Solutions for N=5:")

solve\_n\_queens(5)

print("Solutions for N=8:")

solve\_n\_queens(8)

**Output: Solutions for N=4:**

**.Q..**

**...Q**

**Q...**

**..Q.**

**..Q.**

**Q...**

**...Q**

**.Q..**

**2.** def solve(board, obs=None):

rows, cols, sols = len(board), len(board[0]), []

obs = obs or []

safe = lambda r, c, qs: (r, c) not in obs and all(c != qc and abs(r - qr) != abs(c - qc) for qr, qc in qs)

def find(r, qs):

if r == rows: sols.append(qs[:]); return

for c in range(cols):

if board[r][c] == '0' and safe(r, c, qs):

qs.append((r, c)); find(r + 1, qs); qs.pop()

find(0, []); return sols

def show(board, qs):

for r in range(len(board)): print(" ".join("Q" if (r, c) in qs else board[r][c] for c in range(len(board[0])))); print()

b8 = [['0'] \* 10 for \_ in range(8)]; s8 = solve(b8)

print("8x10:", len(s8) and show(b8, s8[0]))

b5 = [['0'] \* 5 for \_ in range(5)]; o5 = [(1, 2), (2, 0)]; s5 = solve(b5, o5)

print("5x5 obs:", len(s5) and show(b5, s5[0]))

b6 = [['0'] \* 6 for \_ in range(6)]; r6 = [(1, 2), (2, 3)]; s6 = solve(b6, r6)

print("6x6 res:", len(s6) and show(b6, s6[0]))

**Output: Q . . . .**

**. . R . .**

**. Q . R .**

**. . . . .**

**R . Q . .**

**. . . Q .**

**3.** def solve(b):

def find(b):

for i in range(9):

for j in range(9):

if b[i][j] == '.': return i, j

return None

def valid(b, n, p):

r, c = p

for i in range(9):

if (b[r][i] == n and i != c) or (b[i][c] == n and i != r) or (b[(r // 3) \* 3 + i // 3][(c // 3) \* 3 + i % 3] == n and ((r // 3) \* 3 + i // 3, (c // 3) \* 3 + i % 3) != (r, c)): return False

return True

def go():

f = find(b)

if not f: return True

r, c = f

for n in map(str, range(1, 10)):

if valid(b, n, (r, c)):

b[r][c] = n

if go(): return True

b[r][c] = '.'

return False

go()

**Output: [["5","3","4","6","7","8","9","1","2"], ["6","7","2","1","9","5","3","4","8"], ["1","9","8","3","4","2","5","6","7"], ["8","5","9","7","6","1","4","2","3"], ["4","2","6","8","5","3","7","9","1"], ["7","1","3","9","2","4","8","5","6"], ["9","6","1","5","3","7","2","8","4"], ["2","8","7","4","1","9","6","3","5"], ["3","4","5","2","8","6","1","7","9"]].**

**4.** def solve\_sudoku(b):

def find\_empty(b):

for r in range(9):

for c in range(9):

if b[r][c] == '.': return r, c

return None

def is\_valid(b, n, p):

r, c = p

return all(b[r][i] != n and b[i][c] != n and b[(r // 3) \* 3 + i // 3][(c // 3) \* 3 + i % 3] != n for i in range(9))

def backtrack():

f = find\_empty(b)

if not f: return True

r, c = f

for n in map(str, range(1, 10)):

if is\_valid(b, n, (r, c)):

b[r][c] = n

if backtrack(): return True

b[r][c] = '.'

return False

backtrack()

# Example usage:

board = [

["5", "3", ".", ".", "7", ".", ".", ".", "."],

["6", ".", ".", "1", "9", "5", ".", ".", "."],

[".", "9", "8", ".", ".", ".", ".", "6", "."],

["8", ".", ".", ".", "6", ".", ".", ".", "3"],

["4", ".", ".", "8", ".", "3", ".", ".", "1"],

["7", ".", ".", ".", "2", ".", ".", ".", "6"],

[".", "6", ".", ".", ".", ".", "2", "8", "."],

[".", ".", ".", "4", "1", "9", ".", ".", "5"],

[".", ".", ".", ".", "8", ".", ".", "7", "9"]

]

solve\_sudoku(board)

for row in board:

print(row)

**Output:**

**['5', '3', '4', '6', '7', '8', '9', '1', '2']**

**['6', '7', '2', '1', '9', '5', '3', '4', '8']**

**['1', '9', '8', '3', '4', '2', '5', '6', '7']**

**['8', '5', '9', '7', '6', '1', '4', '2', '3']**

**['4', '2', '6', '8', '5', '3', '7', '9', '1']**

**['7', '1', '3', '9', '2', '4', '8', '5', '6']**

**['9', '6', '1', '5', '3', '7', '2', '8', '4']**

**['2', '8', '7', '4', '1', '9', '6', '3', '5']**

**['3', '4', '5', '2', '8', '6', '1', '7', '9']**

**5.** def solve(nums, target):

count = 0

def backtrack(i, sum):

nonlocal count

if i == len(nums):

if sum == target: count += 1

return

backtrack(i + 1, sum + nums[i])

backtrack(i + 1, sum - nums[i])

backtrack(0, 0)

return count

**Output: 5**

**6.** def solve(a):

n, l, r, s, mod = len(a), [-1] \* (len(a)), [len(a)] \* len(a), [], 10\*\*9 + 7

for i in range(n):

while s and a[s[-1]] >= a[i]: s.pop()

l[i] = s[-1] if s else -1; s.append(i)

s = []

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

while s and a[s[-1]] > a[i]: s.pop()

r[i] = s[-1] if s else n; s.append(i)

return sum((i - l[i]) \* (r[i] - i) \* a[i] for i in range(n)) % mod

**Output: 17**

**7.** def combinationSum(candidates, target):

res = []

def backtrack(start, remaining, path):

if remaining == 0:

res.append(path)

return

for i in range(start, len(candidates)):

if candidates[i] > remaining: break

backtrack(i, remaining - candidates[i], path + [candidates[i]])

candidates.sort()

backtrack(0, target, [])

return res

candidates = [2, 3, 6, 7]

target = 7

print(combinationSum(candidates, target))

**Output**: [[2, 2, 3], [7]]

8. def combinationSum2(candidates, target):

result = []

candidates.sort()

def backtrack(combination, remaining, start):

if remaining == 0:

result.append(combination.copy())

return

if remaining < 0:

return

for i in range(start, len(candidates)):

if i > start and candidates[i] == candidates[i - 1]:

continue

combination.append(candidates[i])

backtrack(combination, remaining - candidates[i], i + 1)

combination.pop()

backtrack([], target, 0)

return result

candidates = [10, 1, 2, 7, 6, 1, 5]

target = 8

result = combinationSum2(candidates, target)

print(result)

**Output**: [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]

9. def permutations(nums):

if len(nums) == 0:

return [[]]

result = []

for i in range(len(nums)):

first = nums[i]

rest = nums[:i] + nums[i + 1:]

sub\_permutations = permutations(rest)

for perm in sub\_permutations:

result.append([first] + perm)

return result

**Output:** [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]

10. def solve(n):

n.sort(); ans=[]

def f(path, used):

if len(path) == len(n): ans.append(path[:]); return

for i in range(len(n)):

if used[i] or (i>0 and n[i]==n[i-1] and not used[i-1]): continue

used[i] = True; path.append(n[i])

f(path, used); path.pop(); used[i] = False

f([], [False]\*len(n)); return ans

nums = [1, 1, 2]

result = permuteUnique(nums)

print(result)

**Output:** [[1, 1, 2], [1, 2, 1], [2, 1, 1]]