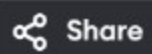




main.py



Run

Output

Clear

```
1 def sum_of_subarray_mins(arr):
2     MOD, n = 10**9 + 7, len(arr)
3     prev_less, next_less, stack = [-1] * n, [n] * n, []
4     for i in range(n):
5         while stack and arr[stack[-1]] >= arr[i]: stack.pop()
6         prev_less[i] = stack[-1] if stack else -1
7         stack.append(i)
8     stack.clear()
9     for i in range(n - 1, -1, -1):
10        while stack and arr[stack[-1]] > arr[i]: stack.pop()
11        next_less[i] = stack[-1] if stack else n
12        stack.append(i)
13    return sum(arr[i] * (i - prev_less[i]) * (next_less[i] - i)
14              for i in range(n)) % MOD
15
16 print(sum_of_subarray_mins([3, 1, 2, 4]))
```

17

=== Code Execution Successful ===

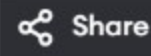


JS

GO



main.py



Run

Output

Clear



JS

GO

```
1 def solve_sudoku(board):
2     def valid(n, r, c):
3         return all(n not in (board[r][i], board[i][c]) for i in
4             range(9)) and \
5             all(n != board[i][j] for i in range(r//3*3, r//3
6                 *3+3) for j in range(c//3*3, c//3*3+3))
7     for r in range(9):
8         for c in range(9):
9             if board[r][c] == '.':
10                 for n in map(str, range(1, 10)):
11                     if valid(n, r, c):
12                         board[r][c] = n
13                         if solve_sudoku(board): return True
14                         board[r][c] = '.'
15                 return False
16     return True
17 board = [
18     ["5","3",".", ".", ".", "7", ".", ".", ".", "."],
19     ["6",".", ".", ".", "1","9","5",".", ".", "."],
20     [".","9","8",".", ".", ".", ".", ".", "6","."],
```

```
['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']
```

=== Code Execution Successful ===



main.py



Share

Run

Output

Clear

```
1 candidates, target = [2, 3, 6, 7], 7
2 res, stack = [], [(0, [], 0)]
3 while stack:
4     start, path, total = stack.pop()
5     if total == target:
6         res.append(path)
7         continue
8     for i in range(start, len(candidates)):
9         if total + candidates[i] <= target:
10             stack.append((i, path + [candidates[i]], total +
                           candidates[i]))
11 print(res)
```

[[7], [2, 2, 3]]

=== Code Execution Successful ===



JS





main.py



Share

Run

Output

Clear



JS

GO

```
1 def solve_sudoku(board):
2     def valid(n, r, c):
3         return all(n not in (board[r][i], board[i][c]) for i in
4             range(9)) and \
5             all(n != board[i][j] for i in range(r//3*3, r//3
6                 *3+3) for j in range(c//3*3, c//3*3+3))
7     for r in range(9):
8         for c in range(9):
9             if board[r][c] == '.':
10                 for n in map(str, range(1, 10)):
11                     if valid(n, r, c):
12                         board[r][c] = n
13                         if solve_sudoku(board): return True
14                         board[r][c] = '.'
15                 return False
16     return True
17 board = [
18     ["5","3",".", ".", ".", "7", ".", ".", ".", "."],
19     ["6",".", ".", ".", "1","9","5",".", ".", "."],
20     [".",".", "9","8",".", ".", ".", ".", "6","."],
```

```
['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']
```

=== Code Execution Successful ===



main.py



Share

Run

Output

Clear



JS



```
1 def solve_n_queens(M, N, obstacles=[], restricted=[]):
2     solutions, board = [], [ "." * N for _ in range(M)]
3     def place(row=0):
4         if row == M:
5             solutions.append("".join(r for r in board))
6             return
7         for col in (restricted if row == 0 and restricted else
8                     range(N)):
9             if (row, col) not in obstacles and all(
10                 board[i][col] != 'Q' and
11                 (col - (row - i) < 0 or board[i][col - (row - i)
12                 ]) != 'Q') and
13                 (col + (row - i) >= N or board[i][col + (row -
14                 i)] != 'Q'):
15                 for i in range(row)):
16                     board[row][col] = 'Q'
17                     place(row + 1)
18                     board[row][col] = '.'
19     place()
20     return solutions
```

Q....

..Q..

....Q

.Q...

...Q.

Q....

...Q.

.Q...

....Q

..Q..

.Q...

...Q.

Q....

..Q..

....Q

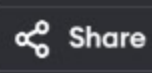
.Q...

....0





main.py



Run

Output

Clear



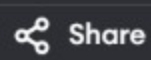
```
1 candidates, target = [10, 1, 2, 7, 6, 1, 5], 8
2 candidates.sort()
3 res, stack = [], [(0, [], 0)]
4 while stack:
5     start, path, total = stack.pop()
6     if total == target:
7         res.append(path)
8         continue
9     for i in range(start, len(candidates)):
10        if i > start and candidates[i] == candidates[i - 1]:
11            continue
12        if total + candidates[i] <= target:
13            stack.append((i + 1, path + [candidates[i]], total +
                           candidates[i]))
14 print(res)
```

```
[[2, 6], [1, 7], [1, 2, 5], [1, 1, 6]]
```

```
=== Code Execution Successful ===
```



main.py



Run

Output

Clear



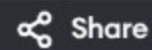
```
10     board[r][c] = n
11     if solve_sudoku(board): return True
12     board[r][c] = '.'
13     return False
14 return True
15 board = [
16     ["5","3",".", ".", ".", "7", ".", ".", ".", "."],
17     ["6",".", ".", ".", "1","9","5",".", ".", ".", "."],
18     [".","9","8",".", ".", ".", ".", ".", "6","."],
19     ["8",".", ".", ".", ".", "6",".", ".", ".", ".", "3"],
20     ["4",".", ".", ".", "8",".", "3",".", ".", ".", "1"],
21     ["7",".", ".", ".", ".", "2",".", ".", ".", ".", "6"],
22     [".","6",".", ".", ".", ".", ".", "2","8","."],
23     [".",".", ".", ".", "4","1","9",".", ".", ".", "5"],
24     [".",".", ".", ".", ".", "8",".", ".", ".", "7","9"]
25 ]
26 solve_sudoku(board)
27 for row in board:
28     print(row)
```

```
['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']
```

=== Code Execution Successful ===



main.py



Run

Output

Clear

```
1  nums = [1, 1, 2]
2  nums.sort()
3  res, stack = [], [(nums, [])]
4  while stack:
5      remaining, path = stack.pop()
6      if not remaining:
7          res.append(path)
8          continue
9      for i in range(len(remaining)):
10         if i > 0 and remaining[i] == remaining[i - 1]:
11             continue
12         stack.append((remaining[:i] + remaining[i + 1:], path +
                        [remaining[i]]))
13  print(res)
```

```
[[2, 1, 1], [1, 2, 1], [1, 1, 2]]
```

```
=== Code Execution Successful ===
```



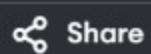
JS

GO





main.py



Run

Output

Clear



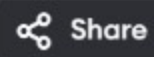
```
1 def find_ways(nums, target):
2     def backtrack(index, total):
3         if index == len(nums):
4             return 1 if total == target else 0
5             return backtrack(index + 1, total + nums[index]) + \
6                 backtrack(index + 1, total - nums[index])
7     return backtrack(0, 0)
8 nums1, target1 = [1, 1, 1, 1, 1], 3
9 print(find_ways(nums1, target1))
```

5

=== Code Execution Successful ===



main.py



Run

Output

Clear

```
5         solutions.append("".join(r) for r in board))
6         return
7     for col in (restricted if row == 0 and restricted else
8                 range(N)):
9         if (row, col) not in obstacles and all(
10             board[i][col] != 'Q' and
11             (col - (row - i) < 0 or board[i][col - (row - i
12                 )] != 'Q') and
13             (col + (row - i) >= N or board[i][col + (row -
14                 i)] != 'Q')
15             for i in range(row)):
16             board[row][col] = 'Q'
17             place(row + 1)
18             board[row][col] = '.'
19         place()
20     return solutions
21
22 for sol in solve_n_queens(5, 5, obstacles=[(1, 1), (3, 3)]):
23     print("\n".join(sol), "\n")
```

Q....

..Q..

....Q

.Q...

...Q.

Q....

...Q.

.Q...

....Q

..Q..

.Q...

...Q.

Q....

..Q..

....Q

.Q...

....Q



main.py



Share

Run

Output

Clear

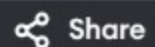
```
10         board[r][c] = n
11         if solve_sudoku(board): return True
12         board[r][c] = '.'
13     return False
14 return True
15 board = [
16     ["5","3",".",".",".","7",".",".","."],
17     ["6",".",".","1","9","5",".","."],
18     [".","9","8",".",".",".","6","."],
19     ["8",".",".","6",".",".","3"],
20     ["4",".",".","8",".","3",".","1"],
21     ["7",".",".","2",".","."],
22     [".","6",".",".","2","8","."],
23     [".",".","4","1","9",".","5"],
24     [".",".","8",".","7","9"]
25 ]
26 solve_sudoku(board)
27 for row in board: print(row)
28
```

```
['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']
```

=== Code Execution Successful ===



main.py



Run

Output

Clear

```
1 N = 4
2 board = [["."] * N for _ in range(N)]
3 solutions = []
4 def solve(row=0):
5     if row == N:
6         solutions.append("".join(r for r in board))
7         return
8     for col in range(N):
9         if all(board[r][col] != 'Q' and
10                (col - (row - r) < 0 or board[r][col - (row - r)
11                ]) != 'Q') and
12                (col + (row - r) >= N or board[r][col + (row - r)
13                ]) != 'Q'):
14                 for r in range(row)):
15                     board[row][col] = 'Q'
16                     solve(row + 1)
17                     board[row][col] = '.'
18 solve()
19 for solution in solutions:
20     print("\n".join(solution), "\n")
```

```
.Q..
...Q
Q...
..Q.

..Q.
Q...
...Q
.Q..
```

```
=== Code Execution Successful ===
```



main.py



Share

Run

Output

Clear

```
1  nums = [1, 2, 3]
2  res, stack = [], [(nums, [])]
3  while stack:
4      remaining, path = stack.pop()
5      if not remaining:
6          res.append(path)
7          continue
8      for i in range(len(remaining)):
9          stack.append((remaining[:i] + remaining[i+1:], path +
                        remaining[i]))
10 print(res)
```

```
[[3, 2, 1], [3, 1, 2], [2, 3, 1], [2, 1, 3], [1, 3, 2], [1, 2, 3]]
```

```
=== Code Execution Successful ===
```