1. Dice throw problem

Program: def dice\_throw(n, m, X):

# Create a table to store results of subproblems

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

# Base case: There is one way to get the sum 0 with 0 dice

dp[0][0] = 1

# Fill the table in bottom-up manner

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

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

dp[i][j] = 0

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

if j - k >= 0:

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

return dp[n][X]

# Example usage:

n = 3 # Number of dice

m = 6 # Number of faces on each die

X = 8 # Desired sum

print(dice\_throw(n, m, X)) # Output: 21

1. Subset sum problem

Program:

def subset\_sum(nums, target):

n = len(nums)

dp = [[False] \* (target + 1) for \_ in range(n + 1)]

# Initialization: a sum of 0 can always be achieved with any number of elements (using the empty subset)

for i in range(n + 1):

dp[i][0] = True

# Fill the DP table

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

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

if j < nums[i - 1]:

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

else:

dp[i][j] = dp[i - 1][j] or dp[i - 1][j - nums[i - 1]]

return dp[n][target]

# Example usage:

nums = [3, 34, 4, 12, 5, 2]

target = 9

print(subset\_sum(nums, target)) # Output: True

1. Assembly line scheduling

Program:

def assembly\_line(a1, a2, t1, t2, e1, e2, x1, x2):

n = len(a1)

T1 = [0] \* n

T2 = [0] \* n

# Initializing base case

T1[0] = e1 + a1[0]

T2[0] = e2 + a2[0]

# Fill tables T1[] and T2[] using the recurrence relations

for i in range(1, n):

T1[i] = min(T1[i-1] + a1[i], T2[i-1] + t2[i-1] + a1[i])

T2[i] = min(T2[i-1] + a2[i], T1[i-1] + t1[i-1] + a2[i])

# Consider exit times

final\_time = min(T1[n-1] + x1, T2[n-1] + x2)

return final\_time

# Example usage

a1 = [4, 5, 3, 2]

a2 = [2, 10, 1, 4]

t1 = [7, 4, 5]

t2 = [9, 2, 8]

e1 = 10

e2 = 12

x1 = 18

x2 = 7

print(assembly\_line(a1, a2, t1, t2, e1, e2, x1, x2)) # Output: 35

1. Longest Palindromic subsequence

Program:

def longest\_palindromic\_subsequence(s):

n = len(s)

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

# Every single character is a palindrome of length 1

for i in range(n):

dp[i][i] = 1

# Build the dp table

for cl in range(2, n + 1): # cl is the length of the substring

for i in range(n - cl + 1):

j = i + cl - 1

if s[i] == s[j] and cl == 2:

dp[i][j] = 2

elif s[i] == s[j]:

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

else:

dp[i][j] = max(dp[i + 1][j], dp[i][j - 1])

return dp[0][n - 1]

# Example usage:

s = "bbbab"

print(longest\_palindromic\_subsequence(s)) # Output: 4