DAY-07

Q01.

num\_sides, num\_dice, target = 6, 2, 7

dp = [[0] \* (target + 1) for \_ in range(num\_dice + 1)]

dp[0][0] = 1

for i in range(1, num\_dice + 1):

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

for k in range(1, num\_sides + 1):

if j >= k:

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

print(dp[num\_dice][target])

output:

6

Q02.

n, a1, a2 = 4, [4, 5, 3, 2], [2, 10, 1, 4]

t1, t2, e1, e2, x1, x2 = [0, 7, 4, 5], [0, 9, 2, 8], 10, 12, 18, 7

dp1, dp2 = [0] \* n, [0] \* n

dp1[0], dp2[0] = e1 + a1[0], e2 + a2[0]

for i in range(1, n):

dp1[i] = min(dp1[i-1] + a1[i], dp2[i-1] + t2[i] + a1[i])

dp2[i] = min(dp2[i-1] + a2[i], dp1[i-1] + t1[i] + a2[i])

print(min(dp1[-1] + x1, dp2[-1] + x2))

output:

35

Q03.

times = [[5, 9, 3], [6, 8, 4], [7, 6, 5]]

transfers = [[0, 2, 3], [2, 0, 4], [3, 4, 0]]

dp = [[float('inf')] \* 3 for \_ in range(3)]

for i in range(3): dp[i][0] = times[i][0]

for j in range(1, 3):

for i in range(3):

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

for k in range(3): dp[i][j] = min(dp[i][j], dp[k][j-1] + transfers[k][i] + times[i][j])

print(min(dp[i][-1] for i in range(3)))

output:

Q04.

import numpy as np

def min\_path\_distance(dist):

n = len(dist)

dp = np.copy(dist)

for k in range(n):

for i in range(n):

for j in range(n):

dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j])

return np.sum(dp) - np.sum(np.diag(dp))

# Test Cases

test\_cases = [

[[0, 10, 15, 20], [10, 0, 35, 25], [15, 35, 0, 30], [20, 25, 30, 0]],

[[0, 10, 10, 10], [10, 0, 10, 10], [10, 10, 0, 10], [10, 10, 10, 0]],

[[0, 1, 2, 3], [1, 0, 4, 5], [2, 4, 0, 6], [3, 5, 6, 0]]

]

for i, case in enumerate(test\_cases):

print(f"Test Case {i+1} Output: {min\_path\_distance(case)}")

ouput:

Test Case 1 Output: 250

Test Case 2 Output: 120

Test Case 3 Output: 36

Q05.

from itertools import permutations

dist = {

'A': {'B': 10, 'C': 15, 'D': 20, 'E': 25},

'B': {'A': 10, 'C': 35, 'D': 25, 'E': 30},

'C': {'A': 15, 'B': 35, 'D': 30, 'E': 20},

'D': {'A': 20, 'B': 25, 'C': 30, 'E': 15},

'E': {'A': 25, 'B': 30, 'C': 20, 'D': 15}

}

cities = list(dist.keys())

min\_dist = float('inf')

best\_route = None

for perm in permutations(cities):

total\_dist = sum(dist[perm[i]][perm[i+1]] for i in range(len(perm)-1)) + dist[perm[-1]][perm[0]]

if total\_dist < min\_dist:

min\_dist = total\_dist

best\_route = perm

print(best\_route, min\_dist)

ouput:

('A', 'B', 'D', 'E', 'C') 85

Q06.

s = "babad"

n = len(s)

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

start, max\_len = 0, 1

for i in range(n): dp[i][i] = True

for i in range(n-1):

if s[i] == s[i+1]: dp[i][i+1] = True; start = i; max\_len = 2

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

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

j = i+length-1

if s[i] == s[j] and dp[i+1][j-1]:

dp[i][j] = True

start = i

max\_len = length

print(s[start:start+max\_len])

ouput:

aba

Q07.

s = "abcabcbb"

char\_map, left, max\_len = {}, 0, 0

for right, ch in enumerate(s):

if ch in char\_map and char\_map[ch] >= left:

left = char\_map[ch] + 1

char\_map[ch] = right

max\_len = max(max\_len, right - left + 1)

print(max\_len)

ouput:

3

Q08.

s, wordDict = "leetcode", ["leet", "code"]

dp = [False] \* (len(s) + 1)

dp[0] = True

for i in range(1, len(s) + 1):

for word in wordDict:

if dp[i - len(word)] and s[i - len(word):i] == word:

dp[i] = True

print(dp[len(s)])

ouput:

True

Q09.

s, wordDict = "ilikesamsung", ["i", "like", "sam", "sung", "samsung"]

dp = [False] \* (len(s) + 1)

dp[0] = True

for i in range(1, len(s) + 1):

for word in wordDict:

if dp[i - len(word)] and s[i - len(word):i] == word:

dp[i] = True

print(dp[len(s)])

ouput:

True

Q10.

words, maxWidth = ["This", "is", "an", "example", "of", "text", "justification."], 16

lines, line, line\_len = [], [], 0

for word in words:

if line\_len + len(word) + len(line) > maxWidth:

lines.append(line)

line, line\_len = [], 0

line.append(word)

line\_len += len(word)

lines.append(line)

for i, line in enumerate(lines):

if i == len(lines) - 1:

print(" ".join(line).ljust(maxWidth))

else:

spaces = maxWidth - sum(len(word) for word in line)

for j in range(spaces):

line[j % (len(line) - 1 or 1)] += ' '

print("".join(line))

ouput:

This is an

example of text

Q11.

class WordFilter:

def \_init\_(self, words):

self.word\_map = {word: i for i, word in enumerate(words)}

def f(self, pref, suff):

for word in reversed(self.word\_map):

if word.startswith(pref) and word.endswith(suff):

return self.word\_map[word]

return -1

wordFilter = WordFilter(["apple"])

print(wordFilter.f("a", "e"))

output: