**Word break problem**

def word\_break(s, wordDict):

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

dp[0] = True

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

for j in range(i):

if dp[j] and s[j:i] in wordDict:

dp[i] = True

break

return dp[len(s)]

# Example Usage

s = "leetcode"

wordDict = ["leet", "code"]

print(word\_break(s, wordDict)) # Output: True

**Word trap problem**

from collections import Counter

def word\_trap(words):

word\_count = Counter(words)

trap\_word = max(word\_count, key=word\_count.get)

return trap\_word

# Example

words = ["apple", "banana", "apple", "cherry", "banana", "apple"]

result = word\_trap(words)

print(result)

**OBST**

def optimal\_bst(keys, freq):

n = len(keys)

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

for i in range(n):

cost[i][i] = freq[i]

for L in range(2, n+1):

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

j = i + L - 1

cost[i][j] = float('inf')

for r in range(i, j+1):

c = sum(freq[i:j+1]) + (cost[i][r-1] if r > i else 0) + (cost[r+1][j] if r < j else 0)

if c < cost[i][j]:

cost[i][j] = c

return cost[0][n-1]

keys = [10, 12, 20]

freq = [34, 8, 50]

print(optimal\_bst(keys, freq))

**Floyd algorithm**

def floyd\_algorithm(graph):

n = len(graph)

dist = graph

for k in range(n):

for i in range(n):

for j in range(n):

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

return dist