Word break

```
def wordBreak(s, wordDict):
    word_set = set(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 word_set:
                dp[i] = True
                break
    return dp[-1]
s = "leetcode"
wordDict = ["leet", "code"]
print(wordBreak(s, wordDict))
```

Word Trap

```
def wordTrap(s, wordDict):
    word_set = set(wordDict)
    n = len(s)
    dp = [False] * (n + 1)
    dp[0] = True
    for i in range(1, n + 1):
        for j in range(i):
            if dp[j] anda s[j:i] in word_set:
                dp[i] = True
                break
    return dp[-1]
s = "applepenapple"
wordDict = ["apple", "pen"]
print(wordTrap(s, wordDict))
```

OBST

```
def optimalBST(keys, freq, n):
    cost = [[0 for x in range(n)] for y 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 = ((0 if r == i else cost[i][r - 1]) +
                     (0 if r == j else cost[r + 1][j]) +
                     sum(freq[i:j + 1]))
                if c < cost[i][j]:
                    cost[i][j] = c

    return cost[0][n - 1]
keys = [10, 12, 20]
freq = [34, 8, 50]
n = len(keys)
print(optimalBST(keys, freq, n))
```

Floyd Algorithm

```
def floydWarshall(graph):
    n = len(graph)
    dist = list(map(lambda i: list(map(lambda j: j, i)), 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
graph = [[0, 5, float('inf'), 10],
         [float('inf'), 0, 3, float('inf')],
         [float('inf'), float('inf'), 0, 1],
         [float('inf'), float('inf'), float('inf'), 0]]

print(floydWarshall(graph))
```