In [1]:

```
#Dice Throw problem
def findWays(faces, dice, sum):
    dp = [[0] * (sum + 1) for _ in range(dice + 1)]
    dp[0][0] = 1

    for i in range(1, dice + 1):
        for j in range(1, sum + 1):
            dp[i][j] = 0
            for k in range(1, faces + 1):
                if j >= k:
                    dp[i][j] += dp[i - 1][j - k]
    return dp[dice][sum]

faces = 6
dice = 3
sum = 8
print(findWays(faces, dice, sum))
```

```
21
```

In [2]:

```
#assembly line scheduling
def carAssembly(a, t, e, x, n):
    T1 = [0] * n
    T2 = [0] * n
    T1[0] = e[0] + a[0][0]
    T2[0] = e[1] + a[1][0]

    for i in range(1, n):
        T1[i] = min(T1[i - 1] + a[0][i], T2[i - 1] + t[1][i] + a[0][i])
        T2[i] = min(T2[i - 1] + a[1][i], T1[i - 1] + t[0][i] + a[1][i])

    return min(T1[n - 1] + x[0], T2[n - 1] + x[1])

# Example
a = [[4, 5, 3, 2], [2, 10, 1, 4]]
t = [[0, 7, 4, 5], [0, 9, 2, 8]]
e = [10, 12]
x = [18, 7]
n = len(a[0])
print(carAssembly(a, t, e, x, n))
```

```
35
```

In [3]:

```
#Travelling salesman problem
def tsp(graph, s):
    n = len(graph)
    dp = [[float('inf')] * (1 << n) for _ in range(n)]
    dp[s][1 << s] = 0

    for mask in range(1 << n):
        for u in range(n):
            if mask & (1 << u):
                for v in range(n):
                    if mask & (1 << v) == 0:
                        dp[v][mask | (1 << v)] = min(dp[v][mask | (1 <<

    return min(dp[i][(1 << n) - 1] + graph[i][s] for i in range(n))

# Example
graph = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]
s = 0
print(tsp(graph, s))
```

```
80
```

In [5]:

```
#Longest palindromic subsequence
def longestPalindromicSubseq(s):
    n = len(s)
    dp = [[0 for _ in range(n)] for _ in range(n)]

    for i in range(n):
        dp[i][i] = 1

    for cl in range(2, n+1):
        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][j-1], dp[i+1][j])

    return dp[0][n-1]

# Example usage:
s = "bbbab"
print(longestPalindromicSubseq(s))
```

```
4
```

In [6]:

```
#Word Break Problem
def wordBreak(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[-1]

# Example usage:
s = "leetcode"
wordDict = ["leet", "code"]
print(wordBreak(s, wordDict))
```

```
True
```

In [7]:

```
#Word Wrap Problem
import sys

def solveWordWrap(nums, k):
    n = len(nums)
    dp = [0 for i in range(n)]
    ans = [0 for i in range(n)]

    for i in range(n-1, -1, -1):
        currlen = -1
        dp[i] = sys.maxsize
        for j in range(i, n):
            currlen += (nums[j] + 1)
            if currlen > k:
                break
            if j == n-1:
                cost = 0
            else:
                cost = ((k-currlen)*(k-currlen)) + dp[j+1]
            if cost < dp[i]:
                dp[i] = cost
                ans[i] = j + 1
    i = 0
    result = []
    while i < n:
        result.append((i+1, ans[i]))
        i = ans[i]

    return result

# Example usage:
nums = [3, 2, 2, 5]
k = 6
print(solveWordWrap(nums, k))
```

```
[(1, 1), (2, 3), (4, 4)]
```

In [ ]:

```
```

In [8]:

```
#Computing a Binomial Coefficient
def binomialCoeff(n, k):
    C = [[0 for x in range(k+1)] for x in range(n+1)]
    for i in range(n+1):
        for j in range(min(i, k)+1):
            if j == 0 or j == i:
                C[i][j] = 1
            else:
                C[i][j] = C[i-1][j-1] + C[i-1][j]
    return C[n][k]

# Example usage:
n = 5
k = 2
print(binomialCoeff(n, k))
```

```
10
```

In [9]:

```
#Floyd-Warshall Algorithm
def floydWarshall(graph):
    V = len(graph)
    dist = list(map(lambda i: list(map(lambda j: j, i)), graph))
    for k in range(V):
        for i in range(V):
            for j in range(V):
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
    return dist

# Example usage:
graph = [[0, 5, float('inf'), 10], [float('inf'), 0, 3, float('inf')],
print(floydWarshall(graph))
```

```
[[0, 5, 8, 9], [inf, 0, 3, 4], [inf, inf, 0, 1], [inf, inf, inf, 0]]
```

In [1]:

```
#Optimal Binary Search Tree
def optimalSearchTree(keys, freq, n):
    dp = [[0 for _ in range(n)] for _ in range(n)]
    sum_freq = [[0 for _ in range(n)] for _ in range(n)]

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

    for L in range(2, n + 1):
        for i in range(n - L + 1):
            j = i + L - 1
            dp[i][j] = float('inf')
            sum_freq[i][j] = sum_freq[i][j - 1] + freq[j]
            for r in range(i, j + 1):
                cost = (dp[i][r - 1] if r > i else 0) + (dp[r + 1][j] i
                if cost < dp[i][j]:
                    dp[i][j] = cost

    return dp[0][n - 1]

# Example usage:
keys = [10, 12, 20]
freq = [34, 8, 50]
n = len(keys)
print(optimalSearchTree(keys, freq, n))
```

```
142
```

In [2]:

```
#knapsack
def knapsack(values, weights, W):
    n = len(values)
    dp = [[0 for _ in range(W + 1)] for _ in range(n + 1)]
    for i in range(1, n + 1):
        for w in range(W + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i -
            else:
                dp[i][w] = dp[i - 1][w]

    return dp[n][W]

values = [60, 100, 120]
weights = [10, 20, 30]
W = 50

max_value = knapsack(values, weights, W)
print(f"The maximum value that can be obtained is {max_value}")
```

```
The maximum value that can be obtained is 220
```

Type *Markdown* and LaTeX: α^2