Computing Binomial Coefficient

```
def binomial_coefficient(n, k):
    if k > n:
        return 0
    if k == 0 or k == n:
        return 1
    dp = [[0] * (k + 1) for _ in range(n + 1)]
    for i in range(n + 1):
        for j in range(min(i, k) + 1):
            if j == 0 or j == i:
                dp[i][j] = 1
            else:
                dp[i][j] = dp[i-1][j-1] + dp[i-1][j]
    return dp[n][k]
n = 5
k = 2
print(binomial_coefficient(n, k))
```

Bellman-Ford Algorithm

```
class Edge:
    def __init__(self, src, dest, weight):
        self.src = src
        self.dest = dest
        self.weight = weight
def bellman_ford(vertices, edges, source):
    dist = [float('inf')] * vertices
    dist[source] = 0
    for _ in range(vertices - 1):
        for edge in edges:
            if dist[edge.src] + edge.weight < dist[edge.dest]:
```

```
            dist[edge.dest] = dist[edge.src] + edge.weight
    for edge in edges:
        if dist[edge.src] + edge.weight < dist[edge.dest]:
            print("Graph contains negative weight cycle")
            return None
    return dist
edges = [
    Edge(0, 1, -1),
    Edge(0, 2, 4),
    Edge(1, 2, 3),
    Edge(1, 3, 2),
    Edge(1, 4, 2),
    Edge(3, 2, 5),
    Edge(3, 1, 1),
    Edge(4, 3, -3)
]
vertices = 5
source = 0
print(bellman_ford(vertices, edges, source))
```

Floyd-Warshall Algorithm

```
def floyd_warshall(graph):
    num_vertices = len(graph)
    dist = list(map(lambda i: list(map(lambda j: j, i)), graph))
    for k in range(num_vertices):
        for i in range(num_vertices):
            for j in range(num_vertices):
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
    return dist
INF = float('inf')
graph = [
```

```
    [0, 3, INF, INF],
    [2, 0, INF, INF],
    [INF, 7, 0, 1],
    [6, INF, INF, 0]
]
print(floyd_warshall(graph))
```

Meet in the Middle Technique

```
def subset_sum(arr, target):
    n = len(arr)
    left = arr[:n//2]
    right = arr[n//2:]
    def generate_subsets(arr):
        subsets = []
        n = len(arr)
        for i in range(1 << n):
            subset_sum = 0
            for j in range(n):
                if i & (1 << j):
                    subset_sum += arr[j]
            subsets.append(subset_sum)
        return subsets
    left_subsets = generate_subsets(left)
    right_subsets = generate_subsets(right)
    right_subsets.sort()
    for subset_sum in left_subsets:
        if binary_search(right_subsets, target - subset_sum):
            return True
    return False
def binary_search(arr, x):
```

```
    lo, hi = 0, len(arr) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if arr[mid] == x:
            return True
        elif arr[mid] < x:
            lo = mid + 1
        else:
            hi = mid - 1
    return False
arr = [3, 34, 4, 12, 5, 2]
target = 9
print(subset_sum(arr, target))
```