**Computing binomial coefficient**

import math

def binomial\_coefficient(n, k):

return math.factorial(n) // (math.factorial(k) \* math.factorial(n - k))

n = 5

k = 2

result = binomial\_coefficient(n, k)

print(f"The binomial coefficient of ({n}, {k}) is: {result}")

**. Bellman ford**

def bellman\_ford(graph, source):

distance = {node: float('infinity') for node in graph}

distance[source] = 0

for \_ in range(len(graph) - 1):

for u, v, weight in graph.edges(data='weight'):

if distance[u] + weight < distance[v]:

distance[v] = distance[u] + weight

for u, v, weight in graph.edges(data='weight'):

if distance[u] + weight < distance[v]:

print("Graph contains negative weight cycle")

return distance

**Warshal Floyd**

def floyd\_warshall(graph):

n = len(graph)

dist = [[float('inf') for \_ in range(n)] for \_ in range(n)]

for i in range(n):

for j in range(n):

dist[i][j] = graph[i][j]

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

# Example graph

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]

]

result = floyd\_warshall(graph)

for row in result:

print(row)

**Meet in the middle technique**

def meet\_in\_the\_middle(target, nums):

result = []

for i in range(1 << len(nums)):

subset = [nums[j] for j in range(len(nums)) if (i & (1 << j))]

if sum(subset) == target:

result.append(subset)

return result

target\_sum = 9

numbers = [3, 1, 4, 6, 5, 2]

print(meet\_in\_the\_middle(target\_sum, numbers))