```
def binomial_coefficient(n, k):
    if k > n:
        return 0
    if k == 0 or k == n:
        return 1
C = [0] * (k + 1)
    C[0] = 1
 for i in range(1, n + 1):
        j = min(i, k)
        while j > 0:
            C[j] = C[j] + C[j - 1]
            j -= 1
         return C[k]
n = 5
k = 2
print(f"Binomial Coefficient C({n}, {k}) is {binomial_coefficient(n, k)}")


class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []
    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])
    def bellman_ford(self, src):
        dist = [float("Inf")] * self.V
        dist[src] = 0
      for _ in range(self.V - 1):
            for u, v, w in self.graph:
                if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                    dist[v] = dist[u] + w
         for u, v, w in self.graph:
```

```
            if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                print("Graph contains negative weight cycle")
                return
        self.print_solution(dist)
    def print_solution(self, dist):
        print("Vertex Distance from Source")
        for i in range(self.V):
            print(f"{i}\t\t{dist[i]}")
g = Graph(5)
g.add_edge(0, 1, -1)
g.add_edge(0, 2, 4)
g.add_edge(1, 2, 3)
g.add_edge(1, 3, 2)
g.add_edge(1, 4, 2)
g.add_edge(3, 2, 5)
g.add_edge(3, 1, 1)
g.add_edge(4, 3, -3)
g.bellman_ford(0)

def floyd_warshall(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])
    print_solution(dist)
def print_solution(dist):
    print("Shortest distances between every pair of vertices:")
    for i in range(len(dist)):
        for j in range(len(dist[i])):
```

```
            if dist[i][j] == float("Inf"):
                print("Inf", end="\t")
            else:
                print(dist[i][j], end="\t")
        print()
graph = [
    [0, 3, float("Inf"), 5],
    [2, 0, float("Inf"), 4],
    [float("Inf"), 1, 0, float("Inf")],
    [float("Inf"), float("Inf"), 2, 0]
]
floyd_warshall(graph)

def meet_in_the_middle(arr, target):
    n = len(arr)
    X = arr[:n//2]
    Y = arr[n//2:]

    X_sums = set()
    Y_sums = set()

    def generate_subsets_sums(arr):
        n = len(arr)
        sums = set()
        for i in range(1 << n):
            s = 0
            for j in range(n):
                if i & (1 << j):
                    s += arr[j]
            sums.add(s)
        return sums
```

```
    X_sums = generate_subsets_sums(X)
    Y_sums = generate_subsets_sums(Y)
  for x in X_sums:
        if (target - x) in Y_sums:
       return False
arr = [1, 3, 2, 7, 4]
target = 10
print(f"Can target {target} be formed: {meet_in_the_middle(arr, target)}")
```