main.py Output

```python
def floyds_algorithm(n, edges, distanceThreshold):
    INF = float('inf')
    dist = [[INF for _ in range(n)] for _ in range(n)]
    for i in range(n):
        dist[i][i] = 0
    for u, v, w in edges:
        dist[u][v] = w
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
    shortest_path_count = sum(1 for row in dist for d in row if d <= distanceThreshold)
    return shortest_path_count
n = 4
edges = [[0, 1, 3], [1, 2, 1], [1, 3, 4], [2, 3, 1]]
distanceThreshold = 4
shortest_path = floyds_algorithm(n, edges, distanceThreshold)
print(shortest_path)
```

main.py Output

```
9

=== Code Execution Successful ===
```

main.py Output

```python
INF = 99999
graph = [[0, 5, INF, 10],
         [INF, 0, 3, INF],
         [INF, INF, 0, 1],
         [INF, INF, INF, 0]]
def floyd_warshall(graph):
    n = len(graph)
    for k in range(n):
        for i in range(n):
            for j in range(n):
                graph[i][j] = min(graph[i][j], graph[i][k] + graph[k][j])
floyd_warshall(graph)
print("Router A to Router F =",graph[0][3])
graph[1][3] = INF
graph[3][1] = INF
floyd_warshall(graph)
print("Router A to Router F =",graph[0][3])
```

main.py Output

```
Router A to Router F = 9
Router A to Router F = 9

=== Code Execution Successful ===
```

main.py | Output

```python
INF = 99999
cities = ['A', 'B', 'C', 'D']
distances = [[0, 2, INF, INF],
             [INF, 0, INF, INF],
             [INF, 7, 0, 1],
             [6, INF, INF, 0]]
for k in range(len(cities)):
    for i in range(len(cities)):
        for j in range(len(cities)):
            if distances[i][j] >distances[i][k] + distances[k][j]:
                distances[i][j] = distances[i][k] + distances[k][j]
start_city = 'C'
end_city = 'A'
start_index = cities.index(start_city)
end_index = cities.index(end_city)
shortest_distance = distances[start_index][end_index]
print(f"Shortest path from {start_city} to {end_city} is {shortest_distance}")
```

main.py Output

```
Shortest path from C to A is 7

=== Code Execution Successful ===
```

main.py Output

```python
import numpy as np
def optimal_bst(keys, freq):
    n = len(keys)
    cost = np.zeros((n, n))
    root = np.zeros((n, n))
    for i in range(n):
        cost[i][i] = freq[i]
        root[i][i] = 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 = cost[i][r - 1] if r > i else 0
                c += cost[r + 1][j] if r < j else 0
                c += sum(freq[i:j + 1])
                if c < cost[i][j]:
                    cost[i][j] = c
                    root[i][j] = r
    return cost[0][n - 1]
keys = ['A', 'B', 'C', 'D']
freq = [0.1, 0.2, 0.4, 0.3]
result = optimal_bst(keys, freq)
print(result)
```

main.py Output

```
1.7


=== Code Execution Successful ===
```

main.py Output

```python
def optimal_bst(keys, freq):
    n = len(keys)
    cost = [[0 for _ in range(n)] for _ in range(n)]
    root = [[0 for _ in range(n)] for _ 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 = cost[i][r - 1] if r > i else 0
                c += cost[r + 1][j] if r < j else 0
                c += sum(freq[i:j + 1])
                if c < cost[i][j]:
                    cost[i][j] = c
                    root[i][j] = r
    return cost[0][n - 1]
keys = [10, 12, 16, 21]
freq = [4, 2, 6, 3]
result = optimal_bst(keys, freq)
print(result)
```

main.py | Output

```
26


=== Code Execution Successful ===
```

main.py Output

```python
def cat_mouse_game(graph):
    n = len(graph)
    dp = [[[0]*n for _ in range(n)] for _ in range(2)]
    for i in range(n):
        for j in range(n):
            dp[0][i][j] = 1
            dp[1][i][j] = 2
    for i in range(n):
        dp[0][0][i] = dp[1][0][i] = 2
    for i in range(n):
        dp[0][i][i] = dp[1][i][i] = 0
    for i in range(n):
        dp[0][i][0] = dp[1][i][0] = 1
    for i in range(n):
        dp[0][i][n-1] = dp[1][i][n-1] = 2
    return dp[0][1][2]
graph = [[2,5],[3],[0,4,5],[1,4,5],[2,3],[0,2,3]]
result = cat_mouse_game(graph)
print(result)
```

main.py Output

```
1


=== Code Execution Successful ===
```

```
import math
m, n = 3, 7
print(math.comb(m + n - 2, m - 1))
```

```
28

=== Code Execution Successful ===
```

```python
nums = [1,2,3,1,1,3]
print(sum(nums.count(x) * (nums.count(x) - 1) // 2 for x in set(nums)))
```

```
4

=== Code Execution Successful ===
```

main.py Output

```python
from collections import defaultdict
import heapq
def maxProbability(n, edges, succProb, start, end):
    graph = defaultdict(list)
    for i, (a, b) in enumerate(edges):
        graph[a].append((b, succProb[i]))
        graph[b].append((a, succProb[i]))
    pq = [(-1, start)]
    probs = [0] * n
    probs[start] = 1
    while pq:
        prob, node = heapq.heappop(pq)
        if node == end:
            return -prob
        for nei, nei_prob in graph[node]:
            if -prob * nei_prob > probs[nei]:
                probs[nei] = -prob * nei_prob
                heapq.heappush(pq, (prob * nei_prob, nei))
    return 0
n = 3
edges = [[0, 1], [1, 2], [0, 2]]
succProb = [0.5, 0.5, 0.2]
start = 0
end = 2
print(maxProbability(n, edges, succProb, start, end))
```

main.py Output

```
0.25


=== Code Execution Successful ===
```

```python
n, edges, dist = 4, [[0,1,3],[1,2,1],[1,3,4],[2,3,1]], 4
d = [[float('inf')]*n for _ in range(n)]
for i, j, w in edges: d[i][j] = d[j][i] = w
for k in range(n):
    for i in range(n):
        for j in range(n):
            d[i][j] = min(d[i][j], d[i][k] + d[k][j])
res, min_reach = 0, n
for i in range(n):
    reach = sum(1 for j in range(n) if i != j and d[i][j] <= dist)
    if reach <= min_reach: res, min_reach = i, reach
print(res)
```

```
3

=== Code Execution Successful ===
```

main.py Output

```python
import heapq
times = [[2,1,1],[2,3,1],[3,4,1]]
n = 4
k = 2
graph = {i: [] for i in range(1, n + 1)}
for u, v, w in times:
    graph[u].append((v, w))
dist = {i: float('inf') for i in range(1, n + 1)}
dist[k] = 0
pq = [(0, k)]
while pq:
    d, node = heapq.heappop(pq)
    if d > dist[node]:
        continue
    for neighbor, weight in graph[node]:
        if d + weight < dist[neighbor]:
            dist[neighbor] = d + weight
            heapq.heappush(pq, (dist[neighbor], neighbor))
print(max(dist.values()) if all(d < float('inf') for d in dist.values()) else -1)
```

main.py Output

```
2


=== Code Execution Successful ===
```