4.18 MAXIMUM PROBABILITY PATH IN AN UNDIRECTED GRAPH

Question:

You are given an undirected weighted graph of n nodes (0-indexed), represented by an edge list where edges[i] = [a, b] is an undirected edge connecting the nodes a and b with a probability of success of traversing that edge succProb[i]. Given two nodes start and end, find the path with the maximum probability of success to go from start to end and return its success probability. If there is no path from start to end, return 0. Your answer will be accepted if it differs from the correct answer by at most 1e-5.

AIM

To implement a graph traversal algorithm that computes the path with the highest probability of success between two nodes in an undirected graph.

ALGORITHM

1. Build an adjacency list graph where each node maps to its neighbors and associated probabilities.
2. Use a **max-heap (priority queue)** to always expand the path with the highest current probability.
3. Initialize a probability array probTo[i] to track the best known probability to reach node i.
4. Start from the start node with probability 1.0.
5. While the heap is not empty:
    - Pop the node with the highest probability.
    - For each neighbor, calculate the new probability.
    - If the new probability is better than the current one, update and push to the heap.
6. Return the probability to reach end, or 0 if unreachable

PROGRAM

```
import heapq
from collections import defaultdict

def max_probability(n, edges, succProb, start, end):
    graph = defaultdict(list)
    for (u, v), prob in zip(edges, succProb):
        graph[u].append((v, prob))
        graph[v].append((u, prob))

    prob = [0.0] * n
    prob[start] = 1.0
    heap = [(-1.0, start)]

    while heap:
        curr_prob, node = heapq.heappop(heap)
        curr_prob = -curr_prob
        if node == end:
            return curr_prob
        for neighbor, edge_prob in graph[node]:
            new_prob = curr_prob * edge_prob
            if new_prob > prob[neighbor]:
                prob[neighbor] = new_prob
                heapq.heappush(heap, (-new_prob, neighbor))
    return 0.0

n = int(input("Enter number of nodes: "))
m = int(input("Enter number of edges: "))
edges = []
succProb = []
for _ in range(m):
    a, b = map(int, input("Edge: ").split())
    edges.append([a, b])
    p = float(input("Success probability: "))
    succProb.append(p)
start = int(input("Start node: "))
end = int(input("End node: "))
result = max_probability(n, edges, succProb, start, end)
print("Maximum success probability:", round(result, 5))
```

Input:

n = 3

edges = [[0, 1], [1, 2], [0, 2]]

succProb = [0.5, 0.5, 0.2]

start = 0

end = 2

Output:

```
Enter number of nodes: 3
Enter number of edges: 3
Edge: 0 1
Success probability: 0.5
Edge: 1 2
Success probability: 0.5
Edge: 0 2
Success probability: 0.2
Start node: 0
End node: 2
Maximum success probability: 0.25
>>>
```

RESULT:

Thus the program is successfully executed and the output is verified.

PERFORMANCE ANALYSIS:

- Time Complexity: O(E log N), where E is the number of edges and N is the number of nodes
- Space Complexity: O(N+E), for storing the graph and heap