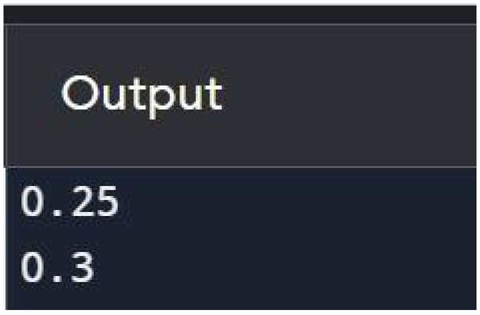
189. You are given an undirected weighted graph of n nodes (0-indexed), represented by an edge list where edges[i] = [a, b] is an undirected edge connecting the nodes a and b with a probability of success of traversing that edge succProb[i]. Given two nodes start and end, find the path with the maximum 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.

Program:

```
from collections import defaultdict
import heapq


def maxProbability(n, edges, succProb, start, end):
    graph = defaultdict(list)
    for i, (a, b) in enumerate(edges):
        prob = succProb[i]
        graph[a].append((b, prob))
        graph[b].append((a, prob))

    pq = [(-1, start)]
    probs = [0] * n
    probs[start] = 1

    while pq:
        cur_prob, node = heapq.heappop(pq)
        cur_prob = -cur_prob

        if node == end:
            return cur_prob

        for neighbor, prob in graph[node]:
            new_prob = cur_prob * prob
            if new_prob > probs[neighbor]:
                probs[neighbor] = new_prob
                heapq.heappush(pq, (-new_prob, neighbor))
    return 0

# Example 1
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))  # Output: 0.25000

# Example 2
n = 3
edges = [[0, 1], [1, 2], [0, 2]]
succProb = [0.5, 0.5, 0.3]
start = 0
end = 2
print(maxProbability(n, edges, succProb, start, end))  # Output: 0.30000
```

Output:



```
0.25
0.3
```

Timecomplexity:O(M*N)