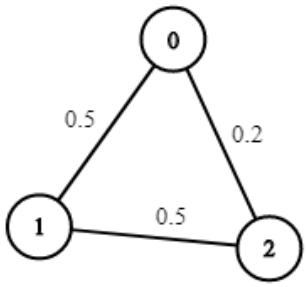
1514. Path with Maximum Probability

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.

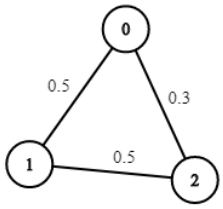
If there is no path from `start` to `end`, **return 0**. Your answer will be accepted if it differs from the correct answer by at most **1e-5**.

Example 1:



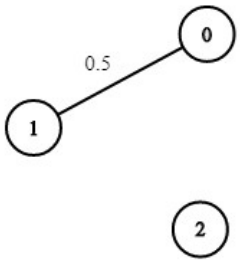

```
Input: n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.2], start = 0, end = 2
Output: 0.25000
Explanation: There are two paths from start to end, one having a probability of success = 0.2 and the other has 0.5 * 0.5 = 0.25.
```

Example 2:




```
Input: n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.3], start = 0, end = 2
Output: 0.30000
```

Example 3:




```
Input: n = 3, edges = [[0,1]], succProb = [0.5], start = 0, end = 2
Output: 0.00000
Explanation: There is no path between 0 and 2.
```

Constraints:

- `2 <= n <= 10^4`
- `0 <= start, end < n`
- `start != end`
- `0 <= a, b < n`
- `a != b`
- `0 <= succProb.length == edges.length <= 2*10^4`
- `0 <= succProb[i] <= 1`
- There is at most one edge between every two nodes.

1514. Path with Maximum Probability

```
/*
   Anti Dijkstra
   Time complexity: O(m+nm)=O(nm)
   Space complexity: O(n+n+nm+2n)=O(n+nm)=O(nm)
   n: #nodes
   m: #edges
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

typedef std::pair<double,int> di;
typedef std::vector<di> vdi;
typedef std::vector<vdi> vvdi;

typedef std::vector<double> vd;

class Solution {
   vvdi graph;
   public:
      void build_graph(int n,vvi& edges, vd& succProb){
         int m=edges.size();
         graph.resize(n);
         for(int i=0;i<m;++i){
            auto edge=edges[i];
            int u= edge[0];
            int v= edge[1];
            double w=succProb[i];
            graph[u].push_back({w,v});
            graph[v].push_back({w,u});
         }
      }
```

```
    double anti_dijkstra(int n,int start, int target){
        vd ans(n,INT_MIN);
        ans[start]=1;
        vi visited(n,false);
        std::priority_queue<di,vdi,std::greater<di>> min_heap;
        min_heap.push({0,start});
        while(!min_heap.empty()){
            di cur=min_heap.top();
            min_heap.pop();
            int u=cur.second;
            if(visited[u]) continue;
            visited[u]=true;
            if(u==target) return ans[u];
            for(auto& edge: graph[u]){
                int v=edge.second;
                double w=edge.first;
                if(ans[v]<=ans[u]*w){
                    ans[v]=ans[u]*w;
                    min_heap.push({1.0/ans[v],v});
                }
            }
        }
        return ans[target]!=INT_MIN?ans[target]:0;
    }

    double maxProbability(int n, vvi& edges, vd& succProb, int start_node, int end_node){
        build_graph(n,edges,succProb);
        return anti_dijkstra(n,start_node,end_node);
    }
};
```