Problem 1514: Path with Maximum Probability

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

,

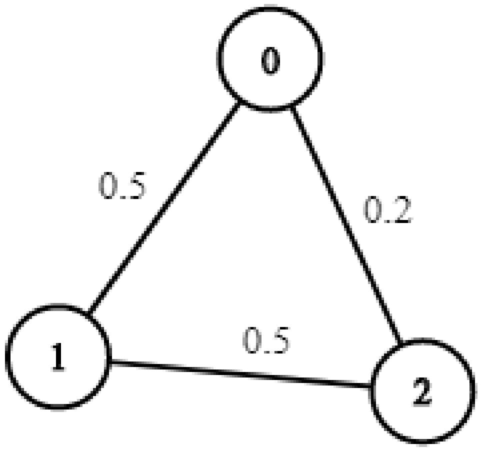
return 0

. 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

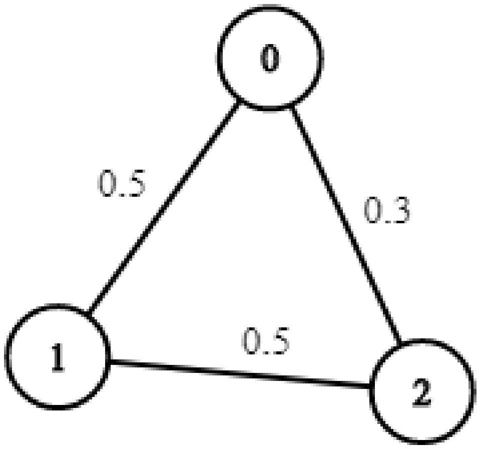
Output:

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:



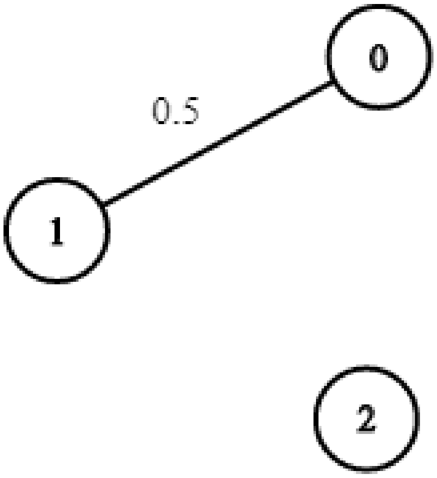
Input:

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.

Code Snippets

C++:

```
class Solution {
public:
double maxProbability(int n, vector<vector<int>>& edges, vector<double>&
succProb, int start_node, int end_node) {

}
};
```

Java:

```
class Solution {
public double maxProbability(int n, int[][] edges, double[] succProb, int
start_node, int end_node) {

}
}
```

Python3:

```
class Solution:
def maxProbability(self, n: int, edges: List[List[int]], succProb:
List[float], start_node: int, end_node: int) -> float:
```

Python:

```
class Solution(object):
def maxProbability(self, n, edges, succProb, start_node, end_node):
"""
:type n: int
:type edges: List[List[int]]
:type succProb: List[float]
:type start_node: int
:type end_node: int
:rtype: float
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number[]} succProb
* @param {number} start_node
* @param {number} end_node
* @return {number}
*/
var maxProbability = function(n, edges, succProb, start_node, end_node) {

};
```

TypeScript:

```
function maxProbability(n: number, edges: number[][], succProb: number[],
start_node: number, end_node: number): number {

};
```

C#:

```
public class Solution {
public double MaxProbability(int n, int[][] edges, double[] succProb, int
start_node, int end_node) {

}
}
```

C:

```
double maxProbability(int n, int** edges, int edgesSize, int* edgesColSize,
double* succProb, int succProbSize, int start_node, int end_node) {

}
```

Go:

```
func maxProbability(n int, edges [][]int, succProb []float64, start_node int,
end_node int) float64 {

}
```

Kotlin:

```
class Solution {
fun maxProbability(n: Int, edges: Array<IntArray>, succProb: DoubleArray,
start_node: Int, end_node: Int): Double {

}
}
```

Swift:

```
class Solution {
func maxProbability(_ n: Int, _ edges: [[Int]], _ succProb: [Double], _
start_node: Int, _ end_node: Int) -> Double {

}
}
```

Rust:

```
impl Solution {
pub fn max_probability(n: i32, edges: Vec<Vec<i32>>, succ_prob: Vec<f64>,
start_node: i32, end_node: i32) -> f64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Float[]} succ_prob
# @param {Integer} start_node
# @param {Integer} end_node
# @return {Float}
def max_probability(n, edges, succ_prob, start_node, end_node)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Float[] $succProb
* @param Integer $start_node
* @param Integer $end_node
* @return Float
*/
function maxProbability($n, $edges, $succProb, $start_node, $end_node) {

}
}
```

Dart:

```
class Solution {
double maxProbability(int n, List<List<int>> edges, List<double> succProb,
int start_node, int end_node) {

}
}
```

Scala:

```
object Solution {
def maxProbability(n: Int, edges: Array[Array[Int]], succProb: Array[Double],
start_node: Int, end_node: Int): Double = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_probability(n :: integer, edges :: [[integer]], succ_prob ::
[float], start_node :: integer, end_node :: integer) :: float
def max_probability(n, edges, succ_prob, start_node, end_node) do

end
end
```

Erlang:

```
-spec max_probability(N :: integer(), Edges :: [[integer()]], SuccProb ::
[float()], Start_node :: integer(), End_node :: integer()) -> float().
max_probability(N, Edges, SuccProb, Start_node, End_node) ->
.
```

Racket:

```
(define/contract (max-probability n edges succProb start_node end_node)
(-> exact-integer? (listof (listof exact-integer?)) (listof flonum?)
exact-integer? exact-integer? flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Path with Maximum Probability
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
double maxProbability(int n, vector<vector<int>>& edges, vector<double>&
succProb, int start_node, int end_node) {

}
};
```

Java Solution:

```
/**
* Problem: Path with Maximum Probability
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public double maxProbability(int n, int[][] edges, double[] succProb, int
start_node, int end_node) {

}
}
```

Python3 Solution:

```
"""
Problem: Path with Maximum Probability
Difficulty: Medium
Tags: array, graph, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maxProbability(self, n: int, edges: List[List[int]], succProb:
List[float], start_node: int, end_node: int) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxProbability(self, n, edges, succProb, start_node, end_node):
"""
:type n: int
:type edges: List[List[int]]
:type succProb: List[float]
:type start_node: int
:type end_node: int
:rtype: float
"""
```

JavaScript Solution:

```
/**
* Problem: Path with Maximum Probability
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @param {number[][]} edges
* @param {number[]} succProb
* @param {number} start_node
* @param {number} end_node
* @return {number}
*/
var maxProbability = function(n, edges, succProb, start_node, end_node) {

};
```

TypeScript Solution:

```
/**
* Problem: Path with Maximum Probability
* Difficulty: Medium
* Tags: array, graph, queue, heap
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maxProbability(n: number, edges: number[][], succProb: number[],
start_node: number, end_node: number): number {

};
```

C# Solution:

```
/*
* Problem: Path with Maximum Probability
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public double MaxProbability(int n, int[][] edges, double[] succProb, int
start_node, int end_node) {

}
}
```

C Solution:

```
/*
* Problem: Path with Maximum Probability
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
double maxProbability(int n, int** edges, int edgesSize, int* edgesColSize,
double* succProb, int succProbSize, int start_node, int end_node) {

}
```

Go Solution:

```
// Problem: Path with Maximum Probability
// Difficulty: Medium
// Tags: array, graph, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maxProbability(n int, edges [][]int, succProb []float64, start_node int,
end_node int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxProbability(n: Int, edges: Array<IntArray>, succProb: DoubleArray,
start_node: Int, end_node: Int): Double {

}
}
```

Swift Solution:

```
class Solution {
func maxProbability(_ n: Int, _ edges: [[Int]], _ succProb: [Double], _
start_node: Int, _ end_node: Int) -> Double {

}
}
```

Rust Solution:

```
// Problem: Path with Maximum Probability
// Difficulty: Medium
// Tags: array, graph, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn max_probability(n: i32, edges: Vec<Vec<i32>>, succ_prob: Vec<f64>,
start_node: i32, end_node: i32) -> f64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Float[]} succ_prob
# @param {Integer} start_node
# @param {Integer} end_node
# @return {Float}
def max_probability(n, edges, succ_prob, start_node, end_node)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Float[] $succProb
* @param Integer $start_node
* @param Integer $end_node
* @return Float
*/
function maxProbability($n, $edges, $succProb, $start_node, $end_node) {

}

}
```

Dart Solution:

```
class Solution {
double maxProbability(int n, List<List<int>> edges, List<double> succProb,
int start_node, int end_node) {

}
}
```

Scala Solution:

```
object Solution {
def maxProbability(n: Int, edges: Array[Array[Int]], succProb: Array[Double],
start_node: Int, end_node: Int): Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_probability(n :: integer, edges :: [[integer]], succ_prob ::
[float], start_node :: integer, end_node :: integer) :: float
def max_probability(n, edges, succ_prob, start_node, end_node) do

end
end
```

Erlang Solution:

```
-spec max_probability(N :: integer(), Edges :: [[integer()]], SuccProb ::
[float()], Start_node :: integer(), End_node :: integer()) -> float().
max_probability(N, Edges, SuccProb, Start_node, End_node) ->
.
```

Racket Solution:

```
(define/contract (max-probability n edges succProb start_node end_node)
(-> exact-integer? (listof (listof exact-integer?)) (listof flonum?)
exact-integer? exact-integer? flonum?)
)
```