Problem 1548: The Most Similar Path in a Graph

Problem Information

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

Problem Description

We have

n

cities and

m

bi-directional

roads

where

roads[i] = [a

i

, b

i

]

connects city

a

i

with city

b

i

. Each city has a name consisting of exactly three upper-case English letters given in the string array

names

. Starting at any city

x

, you can reach any city

y

where

y != x

(i.e., the cities and the roads are forming an undirected connected graph).

You will be given a string array

targetPath

. You should find a path in the graph of the

same length

and with the

minimum edit distance

to

targetPath

.

You need to return

the order of the nodes in the path with the minimum edit distance

. The path should be of the same length of

targetPath

and should be valid (i.e., there should be a direct road between

ans[i]

and

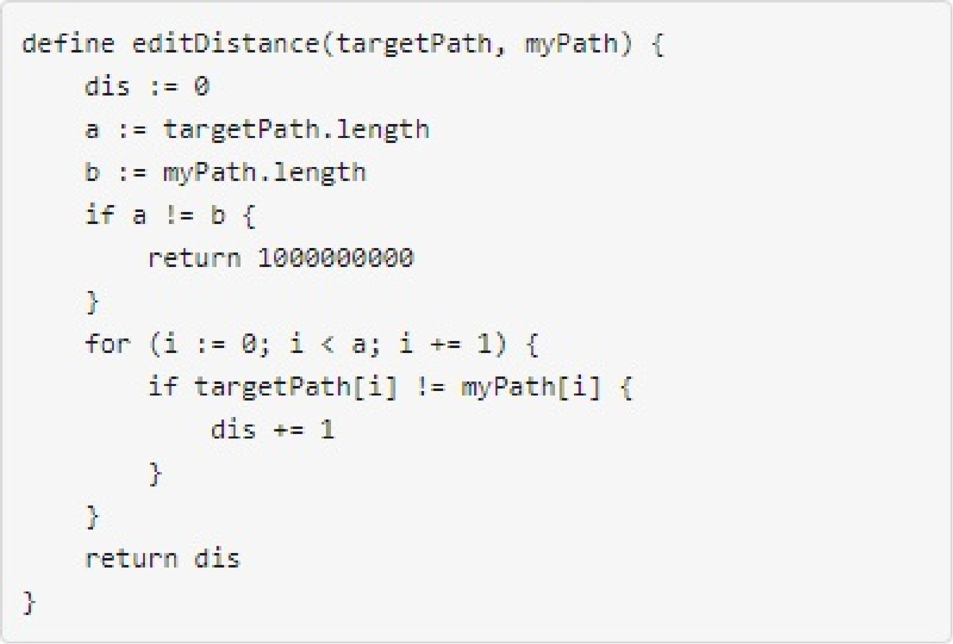
ans[i + 1]

). If there are multiple answers return any one of them.

The

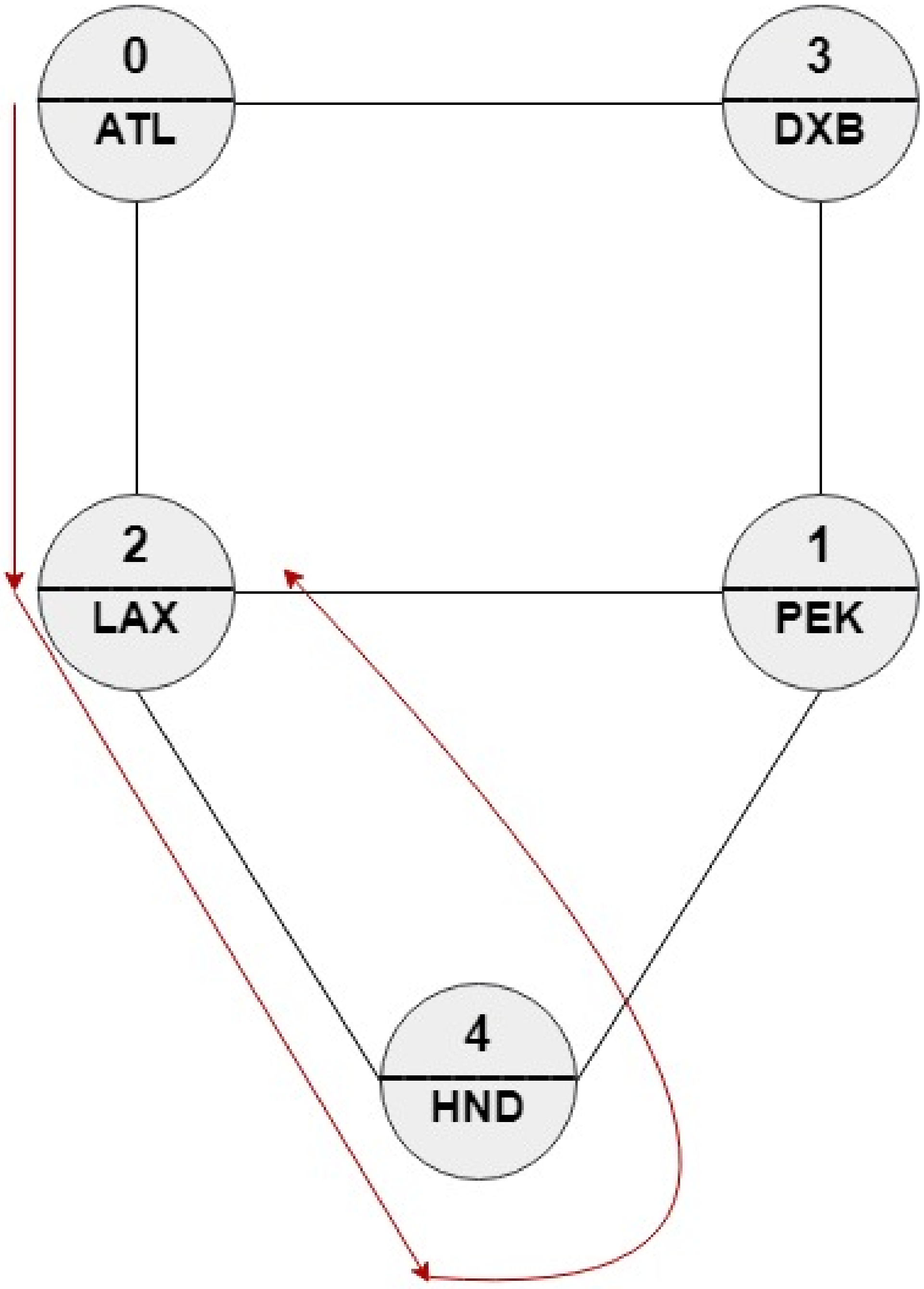
edit distance

is defined as follows:



```
define editDistance(targetPath, myPath) {
    dis := 0
    a := targetPath.length
    b := myPath.length
    if a != b {
        return 1000000000
    }
    for (i := 0; i < a; i += 1) {
        if targetPath[i] != myPath[i] {
            dis += 1
        }
    }
    return dis
}
```

Example 1:



Input:

n = 5, roads = [[0,2],[0,3],[1,2],[1,3],[1,4],[2,4]], names = ["ATL","PEK","LAX","DXB","HND"], targetPath = ["ATL","DXB","HND","LAX"]

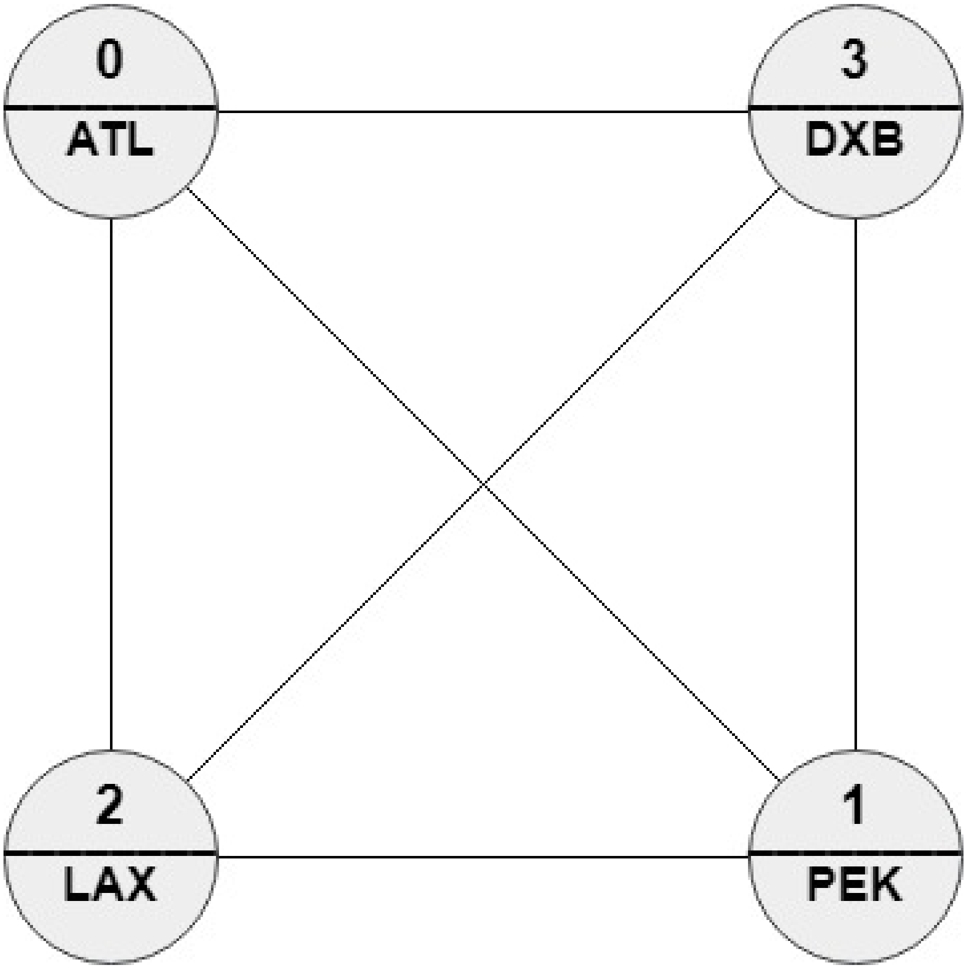
Output:

[0,2,4,2]

Explanation:

[0,2,4,2], [0,3,0,2] and [0,3,1,2] are accepted answers. [0,2,4,2] is equivalent to ["ATL","LAX","HND","LAX"] which has edit distance = 1 with targetPath. [0,3,0,2] is equivalent to ["ATL","DXB","ATL","LAX"] which has edit distance = 1 with targetPath. [0,3,1,2] is equivalent to ["ATL","DXB","PEK","LAX"] which has edit distance = 1 with targetPath.

Example 2:



Input:

n = 4, roads = [[1,0],[2,0],[3,0],[2,1],[3,1],[3,2]], names = ["ATL","PEK","LAX","DXB"], targetPath = ["ABC","DEF","GHI","JKL","MNO","PQR","STU","VWX"]

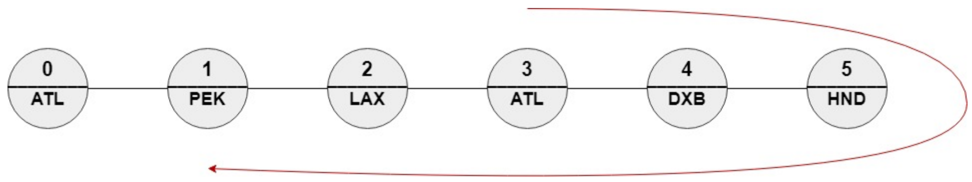
Output:

[0,1,0,1,0,1,0,1]

Explanation:

Any path in this graph has edit distance = 8 with targetPath.

Example 3:



Input:

n = 6, roads = [[0,1],[1,2],[2,3],[3,4],[4,5]], names = ["ATL","PEK","LAX","ATL","DXB","HND"], targetPath = ["ATL","DXB","HND","DXB","ATL","LAX","PEK"]

Output:

[3,4,5,4,3,2,1]

Explanation:

[3,4,5,4,3,2,1] is the only path with edit distance = 0 with targetPath. It's equivalent to ["ATL","DXB","HND","DXB","ATL","LAX","PEK"]

Constraints:

2 <= n <= 100

m == roads.length

n - 1 <= m <= (n * (n - 1) / 2)

0 <= a

i

, b

i

<= n - 1

a

i

!= b

i

The graph is guaranteed to be

connected

and each pair of nodes may have

at most one

direct road.

names.length == n

names[i].length == 3

names[i]

consists of upper-case English letters.

There can be two cities with

the same

name.

1 <= targetPath.length <= 100

targetPath[i].length == 3

targetPath[i]

consists of upper-case English letters.

Follow up:

If each node can be visited only once in the path, What should you change in your solution?

Code Snippets

C++:

```
class Solution {
public:
vector<int> mostSimilar(int n, vector<vector<int>>& roads, vector<string>&
names, vector<string>& targetPath) {

}
};
```

Java:

```
class Solution {
public List<Integer> mostSimilar(int n, int[][] roads, String[] names,
String[] targetPath) {

}
}
```

Python3:

```
class Solution:
def mostSimilar(self, n: int, roads: List[List[int]], names: List[str],
targetPath: List[str]) -> List[int]:
```

Python:

```
class Solution(object):
def mostSimilar(self, n, roads, names, targetPath):
"""
:type n: int
```

```
:type roads: List[List[int]]
:type names: List[str]
:type targetPath: List[str]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} roads
* @param {string[]} names
* @param {string[]} targetPath
* @return {number[]}
*/
var mostSimilar = function(n, roads, names, targetPath) {

};
```

TypeScript:

```
function mostSimilar(n: number, roads: number[][], names: string[],
targetPath: string[]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> MostSimilar(int n, int[][] roads, string[] names, string[]
targetPath) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* mostSimilar(int n, int** roads, int roadsSize, int* roadsColSize, char**
names, int namesSize, char** targetPath, int targetPathSize, int* returnSize)
```

```
{

}
```

Go:

```
func mostSimilar(n int, roads [][]int, names []string, targetPath []string)
[]int {

}
```

Kotlin:

```
class Solution {
fun mostSimilar(n: Int, roads: Array<IntArray>, names: Array<String>,
targetPath: Array<String>): List<Int> {

}
}
```

Swift:

```
class Solution {
func mostSimilar(_ n: Int, _ roads: [[Int]], _ names: [String], _ targetPath:
[String]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn most_similar(n: i32, roads: Vec<Vec<i32>>, names: Vec<String>,
target_path: Vec<String>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} roads
# @param {String[]} names
# @param {String[]} target_path
# @return {Integer[]}
def most_similar(n, roads, names, target_path)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $roads
* @param String[] $names
* @param String[] $targetPath
* @return Integer[]
*/
function mostSimilar($n, $roads, $names, $targetPath) {

}
}
```

Dart:

```
class Solution {
List<int> mostSimilar(int n, List<List<int>> roads, List<String> names,
List<String> targetPath) {

}
}
```

Scala:

```
object Solution {
def mostSimilar(n: Int, roads: Array[Array[Int]], names: Array[String],
targetPath: Array[String]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec most_similar(n :: integer, roads :: [[integer]], names :: [String.t],
target_path :: [String.t]) :: [integer]
def most_similar(n, roads, names, target_path) do

end
end
```

Erlang:

```
-spec most_similar(N :: integer(), Roads :: [[integer()]], Names ::
[unicode:unicode_binary()], TargetPath :: [unicode:unicode_binary()]) ->
[integer()].
most_similar(N, Roads, Names, TargetPath) ->
.
```

Racket:

```
(define/contract (most-similar n roads names targetPath)
(-> exact-integer? (listof (listof exact-integer?)) (listof string?) (listof
string?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Most Similar Path in a Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
vector<int> mostSimilar(int n, vector<vector<int>>& roads, vector<string>&
names, vector<string>& targetPath) {
}
};
```

Java Solution:

```
/**
* Problem: The Most Similar Path in a Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public List<Integer> mostSimilar(int n, int[][] roads, String[] names,
String[] targetPath) {

}
}
```

Python3 Solution:

```
"""
Problem: The Most Similar Path in a Graph
Difficulty: Hard
Tags: array, string, graph, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def mostSimilar(self, n: int, roads: List[List[int]], names: List[str],
targetPath: List[str]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mostSimilar(self, n, roads, names, targetPath):
"""
:type n: int
:type roads: List[List[int]]
:type names: List[str]
:type targetPath: List[str]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: The Most Similar Path in a Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @param {number[][]} roads
* @param {string[]} names
* @param {string[]} targetPath
* @return {number[]}
*/
var mostSimilar = function(n, roads, names, targetPath) {

};
```

TypeScript Solution:

```
/**
* Problem: The Most Similar Path in a Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function mostSimilar(n: number, roads: number[][], names: string[],
targetPath: string[]): number[] {

};
```

C# Solution:

```
/*
* Problem: The Most Similar Path in a Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public IList<int> MostSimilar(int n, int[][] roads, string[] names, string[]
targetPath) {

}
}
```

C Solution:

```
/*
* Problem: The Most Similar Path in a Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

```
* Note: The returned array must be malloced, assume caller calls free().
*/
int* mostSimilar(int n, int** roads, int roadsSize, int* roadsColSize, char**
names, int namesSize, char** targetPath, int targetPathSize, int* returnSize)
{

}
```

Go Solution:

```
// Problem: The Most Similar Path in a Graph
// Difficulty: Hard
// Tags: array, string, graph, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func mostSimilar(n int, roads [][]int, names []string, targetPath []string)
[]int {

}
```

Kotlin Solution:

```
class Solution {
fun mostSimilar(n: Int, roads: Array<IntArray>, names: Array<String>,
targetPath: Array<String>): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func mostSimilar(_ n: Int, _ roads: [[Int]], _ names: [String], _ targetPath:
[String]) -> [Int] {

}
}
```

Rust Solution:

```rust
// Problem: The Most Similar Path in a Graph
// Difficulty: Hard
// Tags: array, string, graph, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn most_similar(n: i32, roads: Vec<Vec<i32>>, names: Vec<String>,
target_path: Vec<String>) -> Vec<i32> {

}
}
```

Ruby Solution:

```ruby
# @param {Integer} n
# @param {Integer[][]} roads
# @param {String[]} names
# @param {String[]} target_path
# @return {Integer[]}
def most_similar(n, roads, names, target_path)

end
```

PHP Solution:

```php
class Solution {

/**
* @param Integer $n
* @param Integer[][] $roads
* @param String[] $names
* @param String[] $targetPath
* @return Integer[]
*/
function mostSimilar($n, $roads, $names, $targetPath) {

}
}
```

Dart Solution:

```
class Solution {
List<int> mostSimilar(int n, List<List<int>> roads, List<String> names,
List<String> targetPath) {

}
}
```

Scala Solution:

```
object Solution {
def mostSimilar(n: Int, roads: Array[Array[Int]], names: Array[String],
targetPath: Array[String]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec most_similar(n :: integer, roads :: [[integer]], names :: [String.t],
target_path :: [String.t]) :: [integer]
def most_similar(n, roads, names, target_path) do

end
end
```

Erlang Solution:

```
-spec most_similar(N :: integer(), Roads :: [[integer()]], Names ::
[unicode:unicode_binary()], TargetPath :: [unicode:unicode_binary()]) ->
[integer()].
most_similar(N, Roads, Names, TargetPath) ->
.
```

Racket Solution:

```
(define/contract (most-similar n roads names targetPath)
(-> exact-integer? (listof (listof exact-integer?)) (listof string?) (listof
string?) (listof exact-integer?))
)
```