Problem 1976: Number of Ways to Arrive at Destination

Problem Information

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

Problem Description

You are in a city that consists of

n

intersections numbered from

0

to

n - 1

with

bi-directional

roads between some intersections. The inputs are generated such that you can reach any intersection from any other intersection and that there is at most one road between any two intersections.

You are given an integer

n

and a 2D integer array

roads

where

roads[i] = [u

i

, v

i

, time

i

]

means that there is a road between intersections

u

i

and

v

i

that takes

time

i

minutes to travel. You want to know in how many ways you can travel from intersection

0

to intersection

n - 1

in the

shortest amount of time

.

Return

the

number of ways

you can arrive at your destination in the

shortest amount of time

. Since the answer may be large, return it

modulo

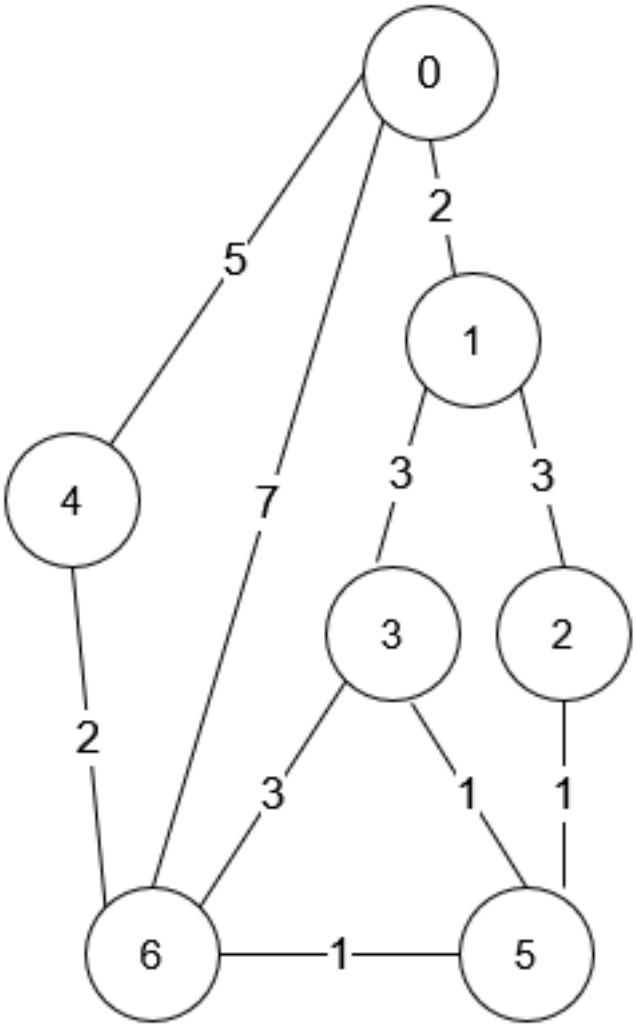
10

9

+ 7

.

Example 1:



Input:

n = 7, roads = [[0,6,7],[0,1,2],[1,2,3],[1,3,3],[6,3,3],[3,5,1],[6,5,1],[2,5,1],[0,4,5],[4,6,2]]

Output:

4

Explanation:

The shortest amount of time it takes to go from intersection 0 to intersection 6 is 7 minutes. The four ways to get there in 7 minutes are: - 0 → 6 - 0 → 4 → 6 - 0 → 1 → 2 → 5 → 6 - 0 → 1 → 3 → 5 → 6

Example 2:

Input:

n = 2, roads = [[1,0,10]]

Output:

1

Explanation:

There is only one way to go from intersection 0 to intersection 1, and it takes 10 minutes.

Constraints:

1 <= n <= 200

n - 1 <= roads.length <= n * (n - 1) / 2

roads[i].length == 3

0 <= u

i

, v

i

<= n - 1

1 <= time

i

<= 10

9

u

i

!= v

i

There is at most one road connecting any two intersections.

You can reach any intersection from any other intersection.

Code Snippets

C++:

```
class Solution {
public:
int countPaths(int n, vector<vector<int>>& roads) {

}
};
```

Java:

```
class Solution {
public int countPaths(int n, int[][] roads) {

}
}
```

Python3:

```
class Solution:
def countPaths(self, n: int, roads: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def countPaths(self, n, roads):
"""
:type n: int
:type roads: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} roads
* @return {number}
*/
var countPaths = function(n, roads) {

};
```

TypeScript:

```
function countPaths(n: number, roads: number[][]): number {

};
```

C#:

```
public class Solution {
public int CountPaths(int n, int[][] roads) {

}
}
```

C:

```
int countPaths(int n, int** roads, int roadsSize, int* roadsColSize) {

}
```

Go:

```
func countPaths(n int, roads [][]int) int {
```

```
}
```

Kotlin:

```
class Solution {
fun countPaths(n: Int, roads: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func countPaths(_ n: Int, _ roads: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_paths(n: i32, roads: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} roads
# @return {Integer}
def count_paths(n, roads)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $roads
```

```
* @return Integer
*/
function countPaths($n, $roads) {

}
}
```

Dart:

```
class Solution {
int countPaths(int n, List<List<int>> roads) {

}
}
```

Scala:

```
object Solution {
def countPaths(n: Int, roads: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_paths(n :: integer, roads :: [[integer]]) :: integer
def count_paths(n, roads) do

end
end
```

Erlang:

```
-spec count_paths(N :: integer(), Roads :: [[integer()]]) -> integer().
count_paths(N, Roads) ->
.
```

Racket:

```
(define/contract (count-paths n roads)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
```

```
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Arrive at Destination
* Difficulty: Medium
* Tags: array, graph, dp, sort
*
* 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:
int countPaths(int n, vector<vector<int>>& roads) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Arrive at Destination
* Difficulty: Medium
* Tags: array, graph, dp, sort
*
* 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 int countPaths(int n, int[][] roads) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Arrive at Destination
Difficulty: Medium
Tags: array, graph, dp, sort

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 countPaths(self, n: int, roads: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countPaths(self, n, roads):
"""
:type n: int
:type roads: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways to Arrive at Destination
* Difficulty: Medium
* Tags: array, graph, dp, sort
*
* 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
```

```
* @return {number}
*/
var countPaths = function(n, roads) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Arrive at Destination
* Difficulty: Medium
* Tags: array, graph, dp, sort
*
* 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 countPaths(n: number, roads: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Arrive at Destination
* Difficulty: Medium
* Tags: array, graph, dp, sort
*
* 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 int CountPaths(int n, int[][] roads) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Arrive at Destination
* Difficulty: Medium
* Tags: array, graph, dp, sort
*
* 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
*/

int countPaths(int n, int** roads, int roadsSize, int* roadsColSize) {

}
```

Go Solution:

```
// Problem: Number of Ways to Arrive at Destination
// Difficulty: Medium
// Tags: array, graph, dp, sort
//
// 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 countPaths(n int, roads [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countPaths(n: Int, roads: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func countPaths(_ n: Int, _ roads: [[Int]]) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Number of Ways to Arrive at Destination
// Difficulty: Medium
// Tags: array, graph, dp, sort
//
// 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 count_paths(n: i32, roads: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} roads
# @return {Integer}
def count_paths(n, roads)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $roads
* @return Integer
*/
function countPaths($n, $roads) {

}
}
```

Dart Solution:

```
class Solution {
int countPaths(int n, List<List<int>> roads) {

}
}
```

Scala Solution:

```
object Solution {
def countPaths(n: Int, roads: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_paths(n :: integer, roads :: [[integer]]) :: integer
def count_paths(n, roads) do

end
end
```

Erlang Solution:

```
-spec count_paths(N :: integer(), Roads :: [[integer()]]) -> integer().
count_paths(N, Roads) ->
.
```

Racket Solution:

```
(define/contract (count-paths n roads)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```