Problem 1334: Find the City With the Smallest Number of Neighbors at a Threshold Distance

Problem Information

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

Problem Description

There are

n

cities numbered from

0

to

n-1

. Given the array

edges

where

edges[i] = [from

i

, to

i

, weight

i

]

represents a bidirectional and weighted edge between cities

from

i

and

to

i

, and given the integer

distanceThreshold

.

Return the city with the smallest number of cities that are reachable through some path and whose distance is

at most

distanceThreshold

, If there are multiple such cities, return the city with the greatest number.

Notice that the distance of a path connecting cities

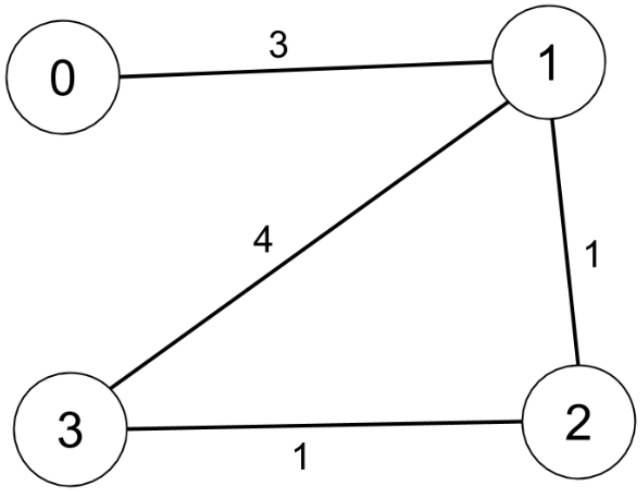
i

and

j

is equal to the sum of the edges' weights along that path.

Example 1:



Input:

n = 4, edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]], distanceThreshold = 4

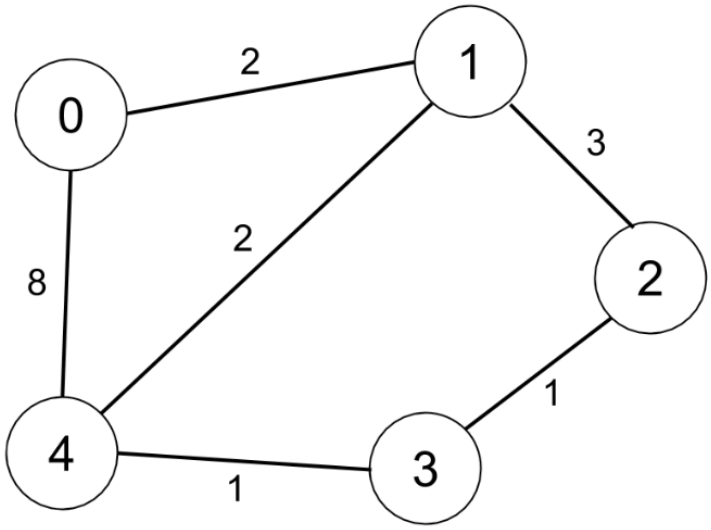
Output:

3

Explanation:

The figure above describes the graph. The neighboring cities at a distanceThreshold = 4 for each city are: City 0 -> [City 1, City 2] City 1 -> [City 0, City 2, City 3] City 2 -> [City 0, City 1, City 3] City 3 -> [City 1, City 2] Cities 0 and 3 have 2 neighboring cities at a distanceThreshold = 4, but we have to return city 3 since it has the greatest number.

Example 2:



Input:

n = 5, edges = [[0,1,2],[0,4,8],[1,2,3],[1,4,2],[2,3,1],[3,4,1]], distanceThreshold = 2

Output:

0

Explanation:

The figure above describes the graph. The neighboring cities at a distanceThreshold = 2 for each city are: City 0 -> [City 1] City 1 -> [City 0, City 4] City 2 -> [City 3, City 4] City 3 -> [City 2, City 4] City 4 -> [City 1, City 2, City 3] The city 0 has 1 neighboring city at a distanceThreshold = 2.

Constraints:

2 <= n <= 100

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

edges[i].length == 3

0 <= from

i

< to

i

< n

1 <= weight

i

, distanceThreshold <= 10^4

All pairs

(from

i

, to

i

)

are distinct.

Code Snippets

C++:

```
class Solution {
public:
int findTheCity(int n, vector<vector<int>>& edges, int distanceThreshold) {
```

```
}
};
```

Java:

```
class Solution {
public int findTheCity(int n, int[][] edges, int distanceThreshold) {

}
}
```

Python3:

```
class Solution:
def findTheCity(self, n: int, edges: List[List[int]], distanceThreshold: int)
-> int:
```

Python:

```
class Solution(object):
def findTheCity(self, n, edges, distanceThreshold):
"""
:type n: int
:type edges: List[List[int]]
:type distanceThreshold: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number} distanceThreshold
* @return {number}
*/
var findTheCity = function(n, edges, distanceThreshold) {

};
```

TypeScript:

```
function findTheCity(n: number, edges: number[][], distanceThreshold:
number): number {

};
```

C#:

```
public class Solution {
public int FindTheCity(int n, int[][] edges, int distanceThreshold) {

}
}
```

C:

```
int findTheCity(int n, int** edges, int edgesSize, int* edgesColSize, int
distanceThreshold) {

}
```

Go:

```
func findTheCity(n int, edges [][]int, distanceThreshold int) int {

}
```

Kotlin:

```
class Solution {
fun findTheCity(n: Int, edges: Array<IntArray>, distanceThreshold: Int): Int
{

}
}
```

Swift:

```
class Solution {
func findTheCity(_ n: Int, _ edges: [[Int]], _ distanceThreshold: Int) -> Int
{

}
}
```

Rust:

```
impl Solution {
pub fn find_the_city(n: i32, edges: Vec<Vec<i32>>, distance_threshold: i32)
-> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} distance_threshold
# @return {Integer}
def find_the_city(n, edges, distance_threshold)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $distanceThreshold
* @return Integer
*/
function findTheCity($n, $edges, $distanceThreshold) {

}
}
```

Dart:

```
class Solution {
int findTheCity(int n, List<List<int>> edges, int distanceThreshold) {

}
}
```

Scala:

```
object Solution {
def findTheCity(n: Int, edges: Array[Array[Int]], distanceThreshold: Int):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_the_city(n :: integer, edges :: [[integer]], distance_threshold ::
integer) :: integer
def find_the_city(n, edges, distance_threshold) do

end
end
```

Erlang:

```
-spec find_the_city(N :: integer(), Edges :: [[integer()]], DistanceThreshold
:: integer()) -> integer().
find_the_city(N, Edges, DistanceThreshold) ->
.
```

Racket:

```
(define/contract (find-the-city n edges distanceThreshold)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the City With the Smallest Number of Neighbors at a Threshold
Distance
* Difficulty: Medium
* Tags: array, 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:
int findTheCity(int n, vector<vector<int>>& edges, int distanceThreshold) {

}
};
```

Java Solution:

```
/**
* Problem: Find the City With the Smallest Number of Neighbors at a Threshold
Distance
* Difficulty: Medium
* Tags: array, 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 int findTheCity(int n, int[][] edges, int distanceThreshold) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the City With the Smallest Number of Neighbors at a Threshold
Distance
Difficulty: Medium
Tags: array, 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 findTheCity(self, n: int, edges: List[List[int]], distanceThreshold: int)
-> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findTheCity(self, n, edges, distanceThreshold):
"""
:type n: int
:type edges: List[List[int]]
:type distanceThreshold: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the City With the Smallest Number of Neighbors at a Threshold
Distance
* Difficulty: Medium
* Tags: array, 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[][]} edges
* @param {number} distanceThreshold
* @return {number}
*/
var findTheCity = function(n, edges, distanceThreshold) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Find the City With the Smallest Number of Neighbors at a Threshold
Distance
* Difficulty: Medium
* Tags: array, 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 findTheCity(n: number, edges: number[][], distanceThreshold:
number): number {

};
```

C# Solution:

```
/*
* Problem: Find the City With the Smallest Number of Neighbors at a Threshold
Distance
* Difficulty: Medium
* Tags: array, 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 int FindTheCity(int n, int[][] edges, int distanceThreshold) {

}
}
```

C Solution:

```
/*
* Problem: Find the City With the Smallest Number of Neighbors at a Threshold
Distance
* Difficulty: Medium
* Tags: array, 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
*/

int findTheCity(int n, int** edges, int edgesSize, int* edgesColSize, int
distanceThreshold) {

}
```

Go Solution:

```
// Problem: Find the City With the Smallest Number of Neighbors at a
Threshold Distance
// Difficulty: Medium
// Tags: array, 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 findTheCity(n int, edges [][]int, distanceThreshold int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findTheCity(n: Int, edges: Array<IntArray>, distanceThreshold: Int): Int
{

}
}
```

Swift Solution:

```
class Solution {
func findTheCity(_ n: Int, _ edges: [[Int]], _ distanceThreshold: Int) -> Int
{

}
}
```

Rust Solution:

```
// Problem: Find the City With the Smallest Number of Neighbors at a
Threshold Distance
// Difficulty: Medium
// Tags: array, 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 find_the_city(n: i32, edges: Vec<Vec<i32>>, distance_threshold: i32)
-> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} distance_threshold
# @return {Integer}
def find_the_city(n, edges, distance_threshold)

end
```

PHP Solution:

```
class Solution {

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

```
* @param Integer $distanceThreshold
* @return Integer
*/
function findTheCity($n, $edges, $distanceThreshold) {

}
}
```

Dart Solution:

```
class Solution {
int findTheCity(int n, List<List<int>> edges, int distanceThreshold) {

}
}
```

Scala Solution:

```
object Solution {
def findTheCity(n: Int, edges: Array[Array[Int]], distanceThreshold: Int):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_the_city(n :: integer, edges :: [[integer]], distance_threshold ::
integer) :: integer
def find_the_city(n, edges, distance_threshold) do

end
end
```

Erlang Solution:

```
-spec find_the_city(N :: integer(), Edges :: [[integer()]], DistanceThreshold
:: integer()) -> integer().
find_the_city(N, Edges, DistanceThreshold) ->
.
```

Racket Solution:

```
(define/contract (find-the-city n edges distanceThreshold)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```