Problem 2045: Second Minimum Time to Reach Destination

Problem Information

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

Problem Description

A city is represented as a

bi-directional connected

graph with

n

vertices where each vertex is labeled from

1

to

n

(

inclusive

). The edges in the graph are represented as a 2D integer array

edges

, where each

edges[i] = [u

i

, v

i

]

denotes a bi-directional edge between vertex

u

i

and vertex

v

i

. Every vertex pair is connected by

at most one

edge, and no vertex has an edge to itself. The time taken to traverse any edge is

time

minutes.

Each vertex has a traffic signal which changes its color from

green

to

red

and vice versa every

change

minutes. All signals change

at the same time

. You can enter a vertex at

any time

, but can leave a vertex

only when the signal is green

. You

cannot wait

at a vertex if the signal is

green

.

The

second minimum value

is defined as the smallest value

strictly larger

than the minimum value.

For example the second minimum value of

[2, 3, 4]

is

3

, and the second minimum value of

[2, 2, 4]

is

4

.

Given

n

,

edges

,

time

, and

change

, return

the

second minimum time

it will take to go from vertex

1

to vertex

n

.

Notes:

You can go through any vertex

any

number of times,

including

1

and

n

.

You can assume that when the journey

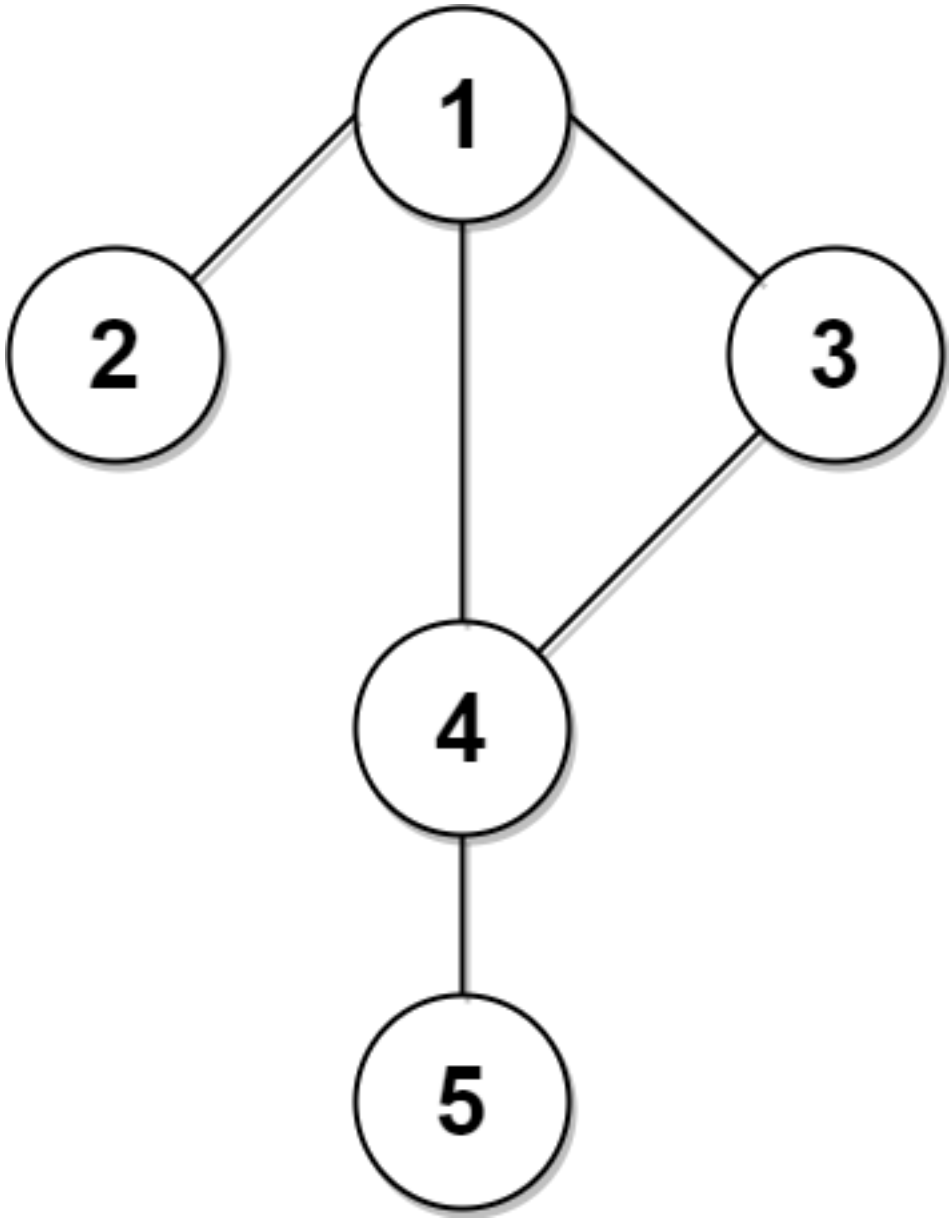
starts

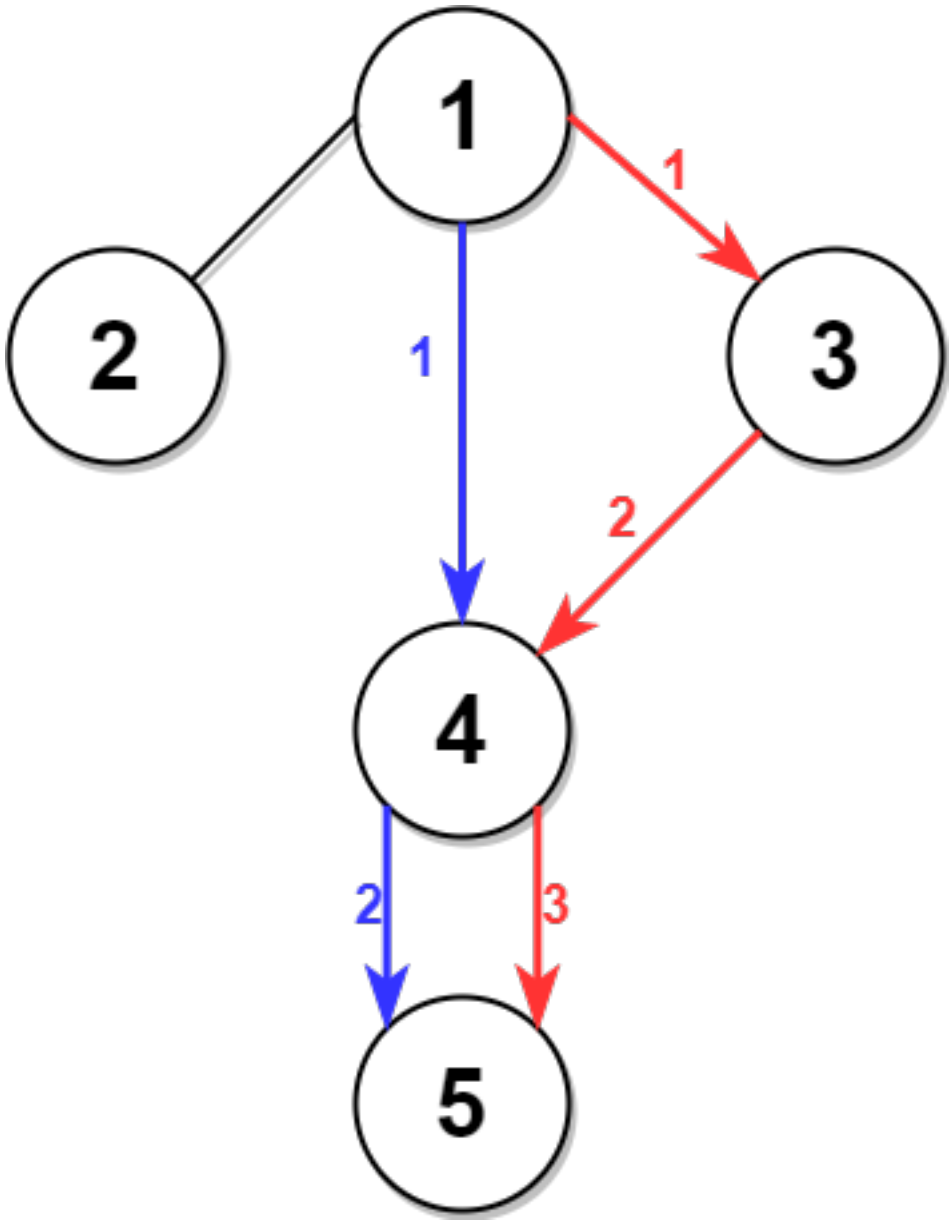
, all signals have just turned

green

.

Example 1:


1
2
3
4
5



Input:

n = 5, edges = [[1,2],[1,3],[1,4],[3,4],[4,5]], time = 3, change = 5

Output:

13

Explanation:

The figure on the left shows the given graph. The blue path in the figure on the right is the minimum time path. The time taken is: - Start at 1, time elapsed=0 - 1 -> 4: 3 minutes, time

elapsed=3 - 4 -> 5: 3 minutes, time elapsed=6 Hence the minimum time needed is 6 minutes.

The red path shows the path to get the second minimum time. - Start at 1, time elapsed=0 - 1 -> 3: 3 minutes, time elapsed=3 - 3 -> 4: 3 minutes, time elapsed=6 - Wait at 4 for 4 minutes, time elapsed=10 - 4 -> 5: 3 minutes, time elapsed=13 Hence the second minimum time is 13 minutes.

Example 2:

Input:

n = 2, edges = [[1,2]], time = 3, change = 2

Output:

11

Explanation:

The minimum time path is 1 -> 2 with time = 3 minutes. The second minimum time path is 1 -> 2 -> 1 -> 2 with time = 11 minutes.

Constraints:

2 <= n <= 10

4

n - 1 <= edges.length <= min(2 * 10

4

, n * (n - 1) / 2)

edges[i].length == 2

1 <= u

i

, v

i

<= n

u

i

!= v

i

There are no duplicate edges.

Each vertex can be reached directly or indirectly from every other vertex.

1 <= time, change <= 10

3

Code Snippets

C++:

```
class Solution {
public:
int secondMinimum(int n, vector<vector<int>>& edges, int time, int change) {

}
};
```

Java:

```
class Solution {
public int secondMinimum(int n, int[][] edges, int time, int change) {

}
}
```

Python3:

```
class Solution:
def secondMinimum(self, n: int, edges: List[List[int]], time: int, change:
int) -> int:
```

Python:

```
class Solution(object):
def secondMinimum(self, n, edges, time, change):
"""
:type n: int
:type edges: List[List[int]]
:type time: int
:type change: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number} time
* @param {number} change
* @return {number}
*/
var secondMinimum = function(n, edges, time, change) {

};
```

TypeScript:

```
function secondMinimum(n: number, edges: number[][], time: number, change:
number): number {
```

```
};
```

C#:

```
public class Solution {
public int SecondMinimum(int n, int[][] edges, int time, int change) {

}
}
```

C:

```
int secondMinimum(int n, int** edges, int edgesSize, int* edgesColSize, int
time, int change) {

}
```

Go:

```
func secondMinimum(n int, edges [][]int, time int, change int) int {

}
```

Kotlin:

```
class Solution {
fun secondMinimum(n: Int, edges: Array<IntArray>, time: Int, change: Int):
Int {

}
}
```

Swift:

```
class Solution {
func secondMinimum(_ n: Int, _ edges: [[Int]], _ time: Int, _ change: Int) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn second_minimum(n: i32, edges: Vec<Vec<i32>>, time: i32, change: i32)
-> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} time
# @param {Integer} change
# @return {Integer}
def second_minimum(n, edges, time, change)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $time
* @param Integer $change
* @return Integer
*/
function secondMinimum($n, $edges, $time, $change) {

}
}
```

Dart:

```
class Solution {
int secondMinimum(int n, List<List<int>> edges, int time, int change) {

}
}
```

Scala:

```
object Solution {
def secondMinimum(n: Int, edges: Array[Array[Int]], time: Int, change: Int):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec second_minimum(n :: integer, edges :: [[integer]], time :: integer,
change :: integer) :: integer
def second_minimum(n, edges, time, change) do

end
end
```

Erlang:

```
-spec second_minimum(N :: integer(), Edges :: [[integer()]], Time ::
integer(), Change :: integer()) -> integer().
second_minimum(N, Edges, Time, Change) ->
.
```

Racket:

```
(define/contract (second-minimum n edges time change)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Second Minimum Time to Reach Destination
* Difficulty: Hard
* Tags: array, graph, search
```

```
*
* 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:
int secondMinimum(int n, vector<vector<int>>& edges, int time, int change) {

}
};
```

Java Solution:

```
/**
* Problem: Second Minimum Time to Reach Destination
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 int secondMinimum(int n, int[][] edges, int time, int change) {

}
}
```

Python3 Solution:

```
"""
Problem: Second Minimum Time to Reach Destination
Difficulty: Hard
Tags: array, graph, search

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 secondMinimum(self, n: int, edges: List[List[int]], time: int, change:
int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def secondMinimum(self, n, edges, time, change):
"""
:type n: int
:type edges: List[List[int]]
:type time: int
:type change: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Second Minimum Time to Reach Destination
* Difficulty: Hard
* Tags: array, graph, search
*
* 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} time
* @param {number} change
* @return {number}
*/
var secondMinimum = function(n, edges, time, change) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Second Minimum Time to Reach Destination
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 secondMinimum(n: number, edges: number[][], time: number, change:
number): number {

};
```

C# Solution:

```
/*
* Problem: Second Minimum Time to Reach Destination
* Difficulty: Hard
* Tags: array, graph, search
*
* 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 int SecondMinimum(int n, int[][] edges, int time, int change) {

}
}
```

C Solution:

```
/*
* Problem: Second Minimum Time to Reach Destination
```

```
* Difficulty: Hard
* Tags: array, graph, search
*
* 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
*/

int secondMinimum(int n, int** edges, int edgesSize, int* edgesColSize, int
time, int change) {

}
```

Go Solution:

```
// Problem: Second Minimum Time to Reach Destination
// Difficulty: Hard
// Tags: array, graph, search
//
// 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 secondMinimum(n int, edges [][]int, time int, change int) int {

}
```

Kotlin Solution:

```
class Solution {
fun secondMinimum(n: Int, edges: Array<IntArray>, time: Int, change: Int):
Int {

}
}
```

Swift Solution:

```
class Solution {
func secondMinimum(_ n: Int, _ edges: [[Int]], _ time: Int, _ change: Int) ->
Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Second Minimum Time to Reach Destination
// Difficulty: Hard
// Tags: array, graph, search
//
// 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 second_minimum(n: i32, edges: Vec<Vec<i32>>, time: i32, change: i32)
-> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} time
# @param {Integer} change
# @return {Integer}
def second_minimum(n, edges, time, change)

end
```

PHP Solution:

```
class Solution {

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

```
* @return Integer
*/
function secondMinimum($n, $edges, $time, $change) {

}
}
```

Dart Solution:

```
class Solution {
int secondMinimum(int n, List<List<int>> edges, int time, int change) {

}
}
```

Scala Solution:

```
object Solution {
def secondMinimum(n: Int, edges: Array[Array[Int]], time: Int, change: Int):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec second_minimum(n :: integer, edges :: [[integer]], time :: integer,
change :: integer) :: integer
def second_minimum(n, edges, time, change) do

end
end
```

Erlang Solution:

```
-spec second_minimum(N :: integer(), Edges :: [[integer()]], Time ::
integer(), Change :: integer()) -> integer().
second_minimum(N, Edges, Time, Change) ->
.
```

Racket Solution:

```
(define/contract (second-minimum n edges time change)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer?)
)
```