# Problem 2477: Minimum Fuel Cost to Report to the Capital

## Problem Information

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

## Problem Description

There is a tree (i.e., a connected, undirected graph with no cycles) structure country network consisting of

n

cities numbered from

0

to

n - 1

and exactly

n - 1

roads. The capital city is city

0

. You are given a 2D integer array

roads

where

roads[i] = [a

i

, b

i

]

denotes that there exists a

bidirectional road

connecting cities

a

i

and

b

i

.

There is a meeting for the representatives of each city. The meeting is in the capital city.

There is a car in each city. You are given an integer

seats

that indicates the number of seats in each car.

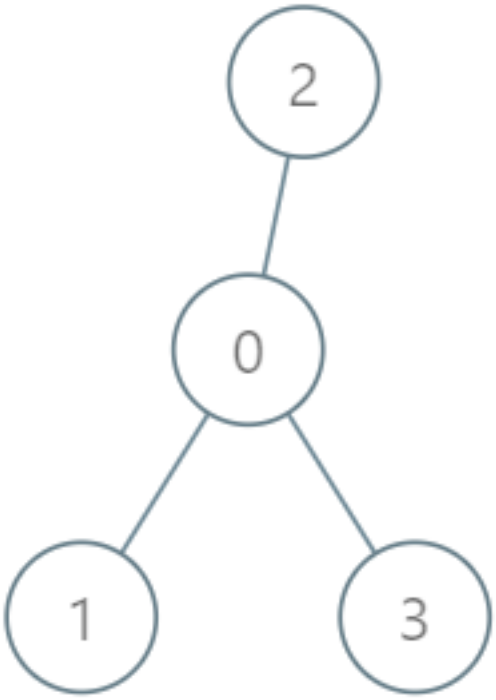
A representative can use the car in their city to travel or change the car and ride with another representative. The cost of traveling between two cities is one liter of fuel.

Return

the minimum number of liters of fuel to reach the capital city

.

Example 1:




Input:

roads = [[0,1],[0,2],[0,3]], seats = 5

Output:

3

Explanation:

- Representative

1

goes directly to the capital with 1 liter of fuel. - Representative

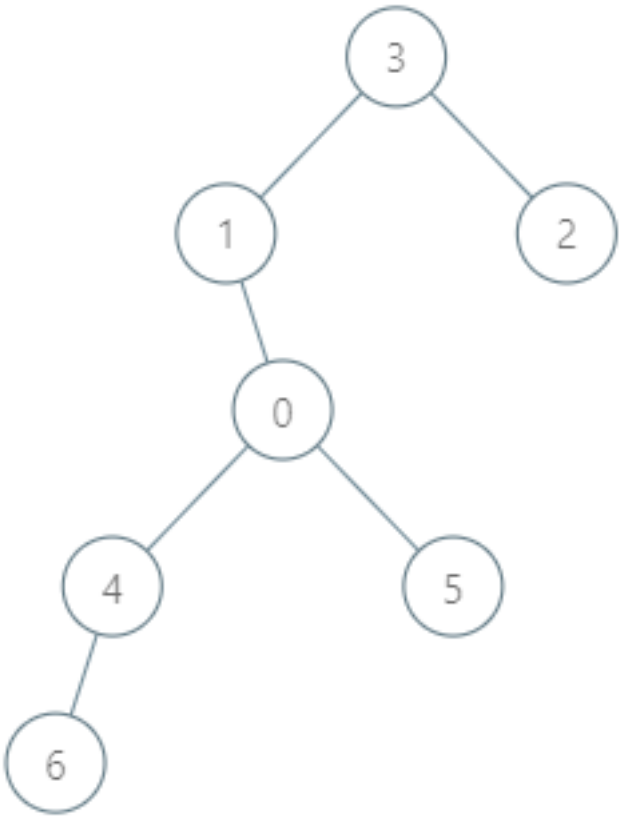
2

goes directly to the capital with 1 liter of fuel. - Representative

3

goes directly to the capital with 1 liter of fuel. It costs 3 liters of fuel at minimum. It can be proven that 3 is the minimum number of liters of fuel needed.

Example 2:



Input:

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

Output:

7

Explanation:

- Representative

2

goes directly to city 3 with 1 liter of fuel. - Representative

2

and representative

3

go together to city 1 with 1 liter of fuel. - Representative

2

and representative

3

go together to the capital with 1 liter of fuel. - Representative

1

goes directly to the capital with 1 liter of fuel. - Representative

5

goes directly to the capital with 1 liter of fuel. - Representative

6

goes directly to city 4 with 1 liter of fuel. - Representative

4

and representative

6

go together to the capital with 1 liter of fuel. It costs 7 liters of fuel at minimum. It can be proven that 7 is the minimum number of liters of fuel needed.

Example 3:


Input:

roads = [], seats = 1

Output:

0

Explanation:

No representatives need to travel to the capital city.

Constraints:

1 <= n <= 10

5

roads.length == n - 1

roads[i].length == 2

0 <= a

i

, b

i

< n

a

i

!= b

i

roads

represents a valid tree.

1 <= seats <= 10

5

## Code Snippets

**C++:**

```
class Solution {
public:
long long minimumFuelCost(vector<vector<int>>& roads, int seats) {
```

```
}
};
```

**Java:**

```
class Solution {
public long minimumFuelCost(int[][] roads, int seats) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function minimumFuelCost(roads: number[][], seats: number): number {

};
```

**C#:**

```
public class Solution {
public long MinimumFuelCost(int[][] roads, int seats) {

}
}
```

**C:**

```
long long minimumFuelCost(int** roads, int roadsSize, int* roadsColSize, int
seats) {

}
```

**Go:**

```
func minimumFuelCost(roads [][]int, seats int) int64 {

}
```

**Kotlin:**

```
class Solution {
fun minimumFuelCost(roads: Array<IntArray>, seats: Int): Long {

}
}
```

**Swift:**

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

}
}
```

**Rust:**

```
impl Solution {
pub fn minimum_fuel_cost(roads: Vec<Vec<i32>>, seats: i32) -> i64 {

}
```

```
}
```

**Ruby:**

```
# @param {Integer[][]} roads
# @param {Integer} seats
# @return {Integer}
def minimum_fuel_cost(roads, seats)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $roads
* @param Integer $seats
* @return Integer
*/
function minimumFuelCost($roads, $seats) {

}
}
```

**Dart:**

```
class Solution {
int minimumFuelCost(List<List<int>> roads, int seats) {

}
}
```

**Scala:**

```
object Solution {
def minimumFuelCost(roads: Array[Array[Int]], seats: Int): Long = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec minimum_fuel_cost(roads :: [[integer]], seats :: integer) :: integer
def minimum_fuel_cost(roads, seats) do

end
end
```

**Erlang:**

```
-spec minimum_fuel_cost(Roads :: [[integer()]], Seats :: integer()) ->
integer().
minimum_fuel_cost(Roads, Seats) ->
.
```

**Racket:**

```
(define/contract (minimum-fuel-cost roads seats)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Minimum Fuel Cost to Report to the Capital
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
long long minimumFuelCost(vector<vector<int>>& roads, int seats) {

}
};
```

### Java Solution:

```
/**
* Problem: Minimum Fuel Cost to Report to the Capital
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public long minimumFuelCost(int[][] roads, int seats) {

}
}
```

### Python3 Solution:

```
"""
Problem: Minimum Fuel Cost to Report to the Capital
Difficulty: Medium
Tags: array, tree, graph, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def minimumFuelCost(self, roads: List[List[int]], seats: int) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

```
"""
```

### JavaScript Solution:

```
/**
* Problem: Minimum Fuel Cost to Report to the Capital
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[][]} roads
* @param {number} seats
* @return {number}
*/
var minimumFuelCost = function(roads, seats) {

};
```

### TypeScript Solution:

```
/**
* Problem: Minimum Fuel Cost to Report to the Capital
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minimumFuelCost(roads: number[][], seats: number): number {

};
```

### C# Solution:

```
/*
* Problem: Minimum Fuel Cost to Report to the Capital
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public long MinimumFuelCost(int[][] roads, int seats) {

}
}
```

### C Solution:

```
/*
* Problem: Minimum Fuel Cost to Report to the Capital
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

long long minimumFuelCost(int** roads, int roadsSize, int* roadsColSize, int
seats) {

}
```

### Go Solution:

```
// Problem: Minimum Fuel Cost to Report to the Capital
// Difficulty: Medium
// Tags: array, tree, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height
```

```
func minimumFuelCost(roads [][]int, seats int) int64 {

}
```

**Kotlin Solution:**

```
class Solution {
fun minimumFuelCost(roads: Array<IntArray>, seats: Int): Long {

}
}
```

**Swift Solution:**

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

}
}
```

**Rust Solution:**

```
// Problem: Minimum Fuel Cost to Report to the Capital
// Difficulty: Medium
// Tags: array, tree, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn minimum_fuel_cost(roads: Vec<Vec<i32>>, seats: i32) -> i64 {

}
}
```

**Ruby Solution:**

```
# @param {Integer[][]} roads
# @param {Integer} seats
```

```
# @return {Integer}
def minimum_fuel_cost(roads, seats)

end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer[][] $roads
* @param Integer $seats
* @return Integer
*/
function minimumFuelCost($roads, $seats) {

}
}
```

**Dart Solution:**

```
class Solution {
int minimumFuelCost(List<List<int>> roads, int seats) {

}
}
```

**Scala Solution:**

```
object Solution {
def minimumFuelCost(roads: Array[Array[Int]], seats: Int): Long = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec minimum_fuel_cost(roads :: [[integer]], seats :: integer) :: integer
def minimum_fuel_cost(roads, seats) do
```

```
end
end
```

### Erlang Solution:

```
-spec minimum_fuel_cost(Roads :: [[integer()]], Seats :: integer()) ->
integer().
minimum_fuel_cost(Roads, Seats) ->
.
```

### Racket Solution:

```
(define/contract (minimum-fuel-cost roads seats)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```