Problem 1168: Optimize Water Distribution in a Village

Problem Information

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

Problem Description

There are

n

houses in a village. We want to supply water for all the houses by building wells and laying pipes.

For each house

i

, we can either build a well inside it directly with cost

wells[i - 1]

(note the

-1

due to

0-indexing

), or pipe in water from another well to it. The costs to lay pipes between houses are given by the array pipes

where each

pipes[j] = [house1

j

, house2

j

, cost

j

]

represents the cost to connect

house1

j

and

house2

j

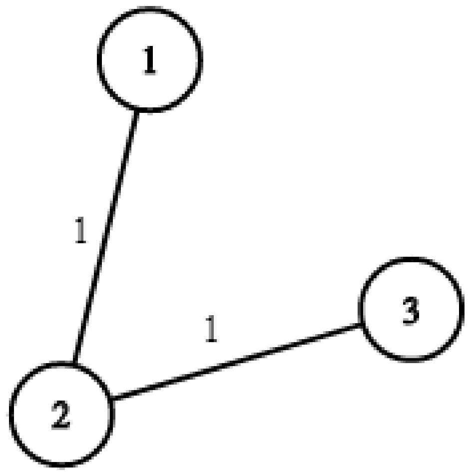
together using a pipe. Connections are bidirectional, and there could be multiple valid connections between the same two houses with different costs.

Return

the minimum total cost to supply water to all houses

.

Example 1:



Input:

n = 3, wells = [1,2,2], pipes = [[1,2,1],[2,3,1]]

Output:

3

Explanation:

The image shows the costs of connecting houses using pipes. The best strategy is to build a well in the first house with cost 1 and connect the other houses to it with cost 2 so the total cost is 3.

Example 2:

Input:

n = 2, wells = [1,1], pipes = [[1,2,1],[1,2,2]]

Output:

2

Explanation:

We can supply water with cost two using one of the three options: Option 1: - Build a well inside house 1 with cost 1. - Build a well inside house 2 with cost 1. The total cost will be 2. Option 2: - Build a well inside house 1 with cost 1. - Connect house 2 with house 1 with cost 1. The total cost will be 2. Option 3: - Build a well inside house 2 with cost 1. - Connect house 1 with house 2 with cost 1. The total cost will be 2. Note that we can connect houses 1 and 2 with cost 1 or with cost 2 but we will always choose

the cheapest option

.

Constraints:

2 <= n <= 10

4

wells.length == n

0 <= wells[i] <= 10

5

1 <= pipes.length <= 10

4

pipes[j].length == 3

1 <= house1

j

, house2

j

<= n

0 <= cost

j

<= 10

5

house1

j

!= house2

j

Code Snippets

C++:

```
class Solution {
public:
int minCostToSupplyWater(int n, vector<int>& wells, vector<vector<int>>&
pipes) {

}
};
```

Java:

```
class Solution {
public int minCostToSupplyWater(int n, int[] wells, int[][] pipes) {

}
}
```

Python3:

```
class Solution:
def minCostToSupplyWater(self, n: int, wells: List[int], pipes:
```

```
List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minCostToSupplyWater(self, n, wells, pipes):
"""
:type n: int
:type wells: List[int]
:type pipes: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[]} wells
* @param {number[][]} pipes
* @return {number}
*/
var minCostToSupplyWater = function(n, wells, pipes) {

};
```

TypeScript:

```
function minCostToSupplyWater(n: number, wells: number[], pipes: number[][]):
number {

};
```

C#:

```
public class Solution {
public int MinCostToSupplyWater(int n, int[] wells, int[][] pipes) {

}
}
```

C:

```
int minCostToSupplyWater(int n, int* wells, int wellsSize, int** pipes, int
pipesSize, int* pipesColSize) {

}
```

Go:

```
func minCostToSupplyWater(n int, wells []int, pipes [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minCostToSupplyWater(n: Int, wells: IntArray, pipes: Array<IntArray>):
Int {

}
}
```

Swift:

```
class Solution {
func minCostToSupplyWater(_ n: Int, _ wells: [Int], _ pipes: [[Int]]) -> Int
{

}
}
```

Rust:

```
impl Solution {
pub fn min_cost_to_supply_water(n: i32, wells: Vec<i32>, pipes:
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} wells
# @param {Integer[][]} pipes
```

```
# @return {Integer}
def min_cost_to_supply_water(n, wells, pipes)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $wells
* @param Integer[][] $pipes
* @return Integer
*/
function minCostToSupplyWater($n, $wells, $pipes) {

}
}
```

Dart:

```
class Solution {
int minCostToSupplyWater(int n, List<int> wells, List<List<int>> pipes) {

}
}
```

Scala:

```
object Solution {
def minCostToSupplyWater(n: Int, wells: Array[Int], pipes:
Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost_to_supply_water(n :: integer, wells :: [integer], pipes ::
[[integer]]) :: integer
```

```
def min_cost_to_supply_water(n, wells, pipes) do

end
end
```

Erlang:

```
-spec min_cost_to_supply_water(N :: integer(), Wells :: [integer()], Pipes ::
[[integer()]]) -> integer().
min_cost_to_supply_water(N, Wells, Pipes) ->
.
```

Racket:

```
(define/contract (min-cost-to-supply-water n wells pipes)
(-> exact-integer? (listof exact-integer?) (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Optimize Water Distribution in a Village
* Difficulty: Hard
* Tags: array, tree, graph, queue, heap
*
* 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:
int minCostToSupplyWater(int n, vector<int>& wells, vector<vector<int>>&
pipes) {

}
};
```

Java Solution:

```
/**
* Problem: Optimize Water Distribution in a Village
* Difficulty: Hard
* Tags: array, tree, graph, queue, heap
*
* 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 int minCostToSupplyWater(int n, int[] wells, int[][] pipes) {

}
}
```

Python3 Solution:

```
"""
Problem: Optimize Water Distribution in a Village
Difficulty: Hard
Tags: array, tree, graph, queue, heap

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 minCostToSupplyWater(self, n: int, wells: List[int], pipes:
List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minCostToSupplyWater(self, n, wells, pipes):
"""
:type n: int
```

```
:type wells: List[int]
:type pipes: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Optimize Water Distribution in a Village
* Difficulty: Hard
* Tags: array, tree, graph, queue, heap
*
* 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} n
* @param {number[]} wells
* @param {number[][]} pipes
* @return {number}
*/
var minCostToSupplyWater = function(n, wells, pipes) {

};
```

TypeScript Solution:

```
/**
* Problem: Optimize Water Distribution in a Village
* Difficulty: Hard
* Tags: array, tree, graph, queue, heap
*
* 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 minCostToSupplyWater(n: number, wells: number[], pipes: number[][]):
number {
```

```
};
```

C# Solution:

```
/*
* Problem: Optimize Water Distribution in a Village
* Difficulty: Hard
* Tags: array, tree, graph, queue, heap
*
* 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 int MinCostToSupplyWater(int n, int[] wells, int[][] pipes) {

}
}
```

C Solution:

```
/*
* Problem: Optimize Water Distribution in a Village
* Difficulty: Hard
* Tags: array, tree, graph, queue, heap
*
* 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
*/

int minCostToSupplyWater(int n, int* wells, int wellsSize, int** pipes, int
pipesSize, int* pipesColSize) {

}
```

Go Solution:

```
// Problem: Optimize Water Distribution in a Village
// Difficulty: Hard
// Tags: array, tree, graph, queue, heap
//
// 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 minCostToSupplyWater(n int, wells []int, pipes [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minCostToSupplyWater(n: Int, wells: IntArray, pipes: Array<IntArray>):
Int {

}
}
```

Swift Solution:

```
class Solution {
func minCostToSupplyWater(_ n: Int, _ wells: [Int], _ pipes: [[Int]]) -> Int
{

}
}
```

Rust Solution:

```
// Problem: Optimize Water Distribution in a Village
// Difficulty: Hard
// Tags: array, tree, graph, queue, heap
//
// 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 min_cost_to_supply_water(n: i32, wells: Vec<i32>, pipes:
```

```
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} wells
# @param {Integer[][]} pipes
# @return {Integer}
def min_cost_to_supply_water(n, wells, pipes)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $wells
* @param Integer[][] $pipes
* @return Integer
*/
function minCostToSupplyWater($n, $wells, $pipes) {

}
}
```

Dart Solution:

```
class Solution {
int minCostToSupplyWater(int n, List<int> wells, List<List<int>> pipes) {

}
}
```

Scala Solution:

```
object Solution {
def minCostToSupplyWater(n: Int, wells: Array[Int], pipes:
```

```
Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost_to_supply_water(n :: integer, wells :: [integer], pipes ::
[[integer]]) :: integer
def min_cost_to_supply_water(n, wells, pipes) do

end
end
```

Erlang Solution:

```
-spec min_cost_to_supply_water(N :: integer(), Wells :: [integer()], Pipes ::
[[integer()]]) -> integer().
min_cost_to_supply_water(N, Wells, Pipes) ->
.
```

Racket Solution:

```
(define/contract (min-cost-to-supply-water n wells pipes)
(-> exact-integer? (listof exact-integer?) (listof (listof exact-integer?))
exact-integer?)
)
```