Problem 3535: Unit Conversion II

Problem Information

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

Problem Description

There are

n

types of units indexed from

0

to

n - 1

.

You are given a 2D integer array

conversions

of length

n - 1

, where

conversions[i] = [sourceUnit

i

, targetUnit

i

, conversionFactor

i

]

. This indicates that a single unit of type

sourceUnit

i

is equivalent to

conversionFactor

i

units of type

targetUnit

i

.

You are also given a 2D integer array

queries

of length

q

, where

queries[i] = [unitA

i

, unitB

i

]

.

Return an array

answer

of length

q

where

answer[i]

is the number of units of type

unitB

i

equivalent to 1 unit of type

unitA

i

, and can be represented as

p/q

where

p

and

q

are coprime. Return each

answer[i]

as

pq

-1

modulo

10

9

+ 7

, where

q

-1

represents the multiplicative inverse of

q

modulo

10

9

+ 7

.

Example 1:

Input:

conversions = [[0,1,2],[0,2,6]], queries = [[1,2],[1,0]]

Output:

[3,500000004]

Explanation:

In the first query, we can convert unit 1 into 3 units of type 2 using the inverse of

conversions[0]

, then

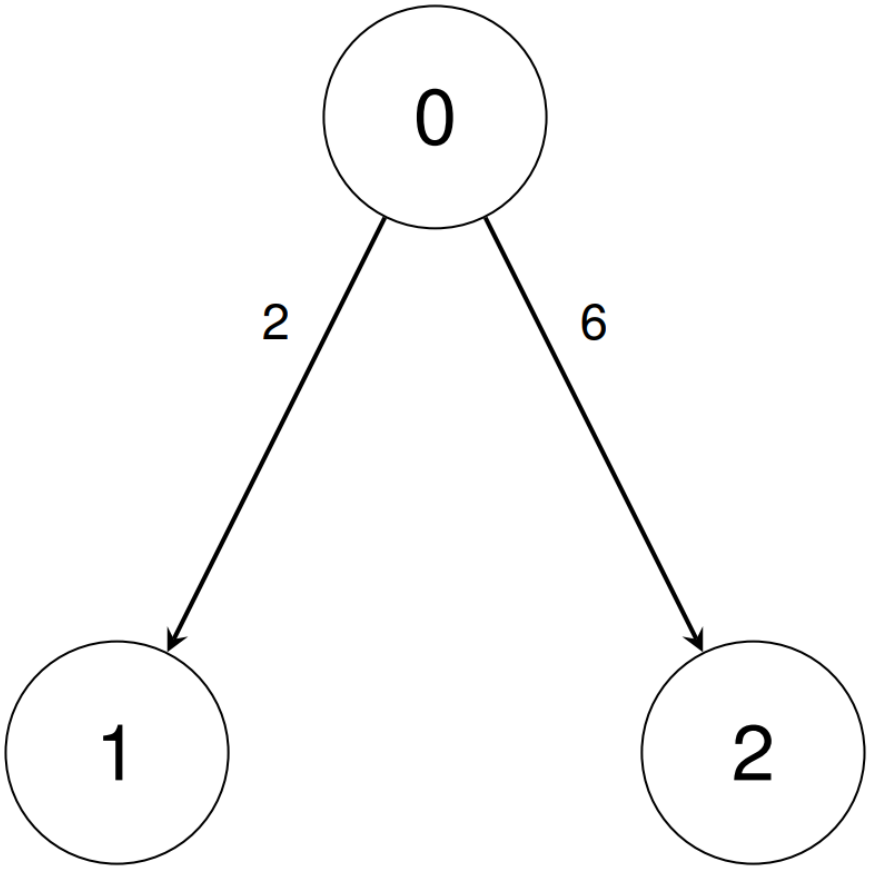
conversions[1]

.

In the second query, we can convert unit 1 into 1/2 units of type 0 using the inverse of

conversions[0]

. We return 500000004 since it is the multiplicative inverse of 2.



Example 2:

Input:

conversions = [[0,1,2],[0,2,6],[0,3,8],[2,4,2],[2,5,4],[3,6,3]], queries = [[1,2],[0,4],[6,5],[4,6],[6,1]]

Output:

[3,12,1,2,83333334]

Explanation:

In the first query, we can convert unit 1 into 3 units of type 2 using the inverse of

conversions[0]

, then

conversions[1]

.

In the second query, we can convert unit 0 into 12 units of type 4 using

conversions[1]

, then

conversions[3]

.

In the third query, we can convert unit 6 into 1 unit of type 5 using the inverse of

conversions[5]

, the inverse of

conversions[2]

,

conversions[1]

, then

conversions[4]

.

In the fourth query, we can convert unit 4 into 2 units of type 6 using the inverse of

conversions[3]

, the inverse of

conversions[1]

,

conversions[2]

, then

conversions[5]

.

In the fifth query, we can convert unit 6 into 1/12 units of type 1 using the inverse of

conversions[5]

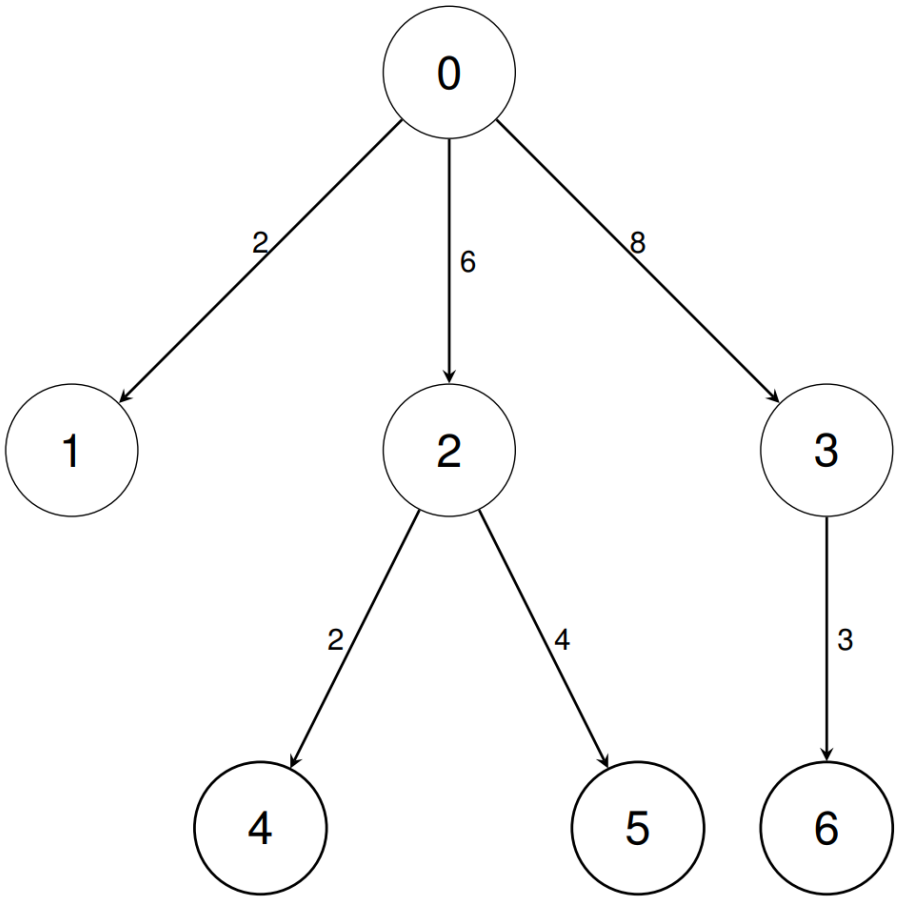
, the inverse of

conversions[2]

, then

conversions[0]

. We return 83333334 since it is the multiplicative inverse of 12.



Constraints:

2 <= n <= 10

5

conversions.length == n - 1

0 <= sourceUnit

i

, targetUnit

i

< n

1 <= conversionFactor

i

<= 10

9

1 <= q <= 10

5

queries.length == q

0 <= unitA

i

, unitB

i

< n

It is guaranteed that unit 0 can be

uniquely

converted into any other unit through a combination of forward or backward conversions.

Code Snippets

C++:

```
class Solution {
public:
vector<int> queryConversions(vector<vector<int>>& conversions,
vector<vector<int>>& queries) {
```

```
}
};
```

Java:

```
class Solution {
public int[] queryConversions(int[][] conversions, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def queryConversions(self, conversions: List[List[int]], queries:
List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def queryConversions(self, conversions, queries):
"""
:type conversions: List[List[int]]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} conversions
* @param {number[][]} queries
* @return {number[]}
*/
var queryConversions = function(conversions, queries) {

};
```

TypeScript:

```
function queryConversions(conversions: number[][], queries: number[][]):
number[] {

};
```

C#:

```
public class Solution {
public int[] QueryConversions(int[][] conversions, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* queryConversions(int** conversions, int conversionsSize, int*
conversionsColSize, int** queries, int queriesSize, int* queriesColSize, int*
returnSize) {

}
```

Go:

```
func queryConversions(conversions [][]int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun queryConversions(conversions: Array<IntArray>, queries: Array<IntArray>):
IntArray {

}
}
```

Swift:

```
class Solution {
func queryConversions(_ conversions: [[Int]], _ queries: [[Int]]) -> [Int] {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn query_conversions(conversions: Vec<Vec<i32>>, queries: Vec<Vec<i32>>)
-> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} conversions
# @param {Integer[][]} queries
# @return {Integer[]}
def query_conversions(conversions, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $conversions
* @param Integer[][] $queries
* @return Integer[]
*/
function queryConversions($conversions, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> queryConversions(List<List<int>> conversions, List<List<int>>
queries) {
```

```
}
}
```

Scala:

```
object Solution {
def queryConversions(conversions: Array[Array[Int]], queries:
Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec query_conversions(conversions :: [[integer]], queries :: [[integer]])
:: [integer]
def query_conversions(conversions, queries) do

end
end
```

Erlang:

```
-spec query_conversions(Conversions :: [[integer()]], Queries ::
[[integer()]]) -> [integer()].
query_conversions(Conversions, Queries) ->
.
```

Racket:

```
(define/contract (query-conversions conversions queries)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Unit Conversion II
* Difficulty: Medium
* Tags: array, graph, math, 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:
vector<int> queryConversions(vector<vector<int>>& conversions,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Unit Conversion II
* Difficulty: Medium
* Tags: array, graph, math, 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[] queryConversions(int[][] conversions, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Unit Conversion II
Difficulty: Medium
Tags: array, graph, math, 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 queryConversions(self, conversions: List[List[int]], queries:
List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def queryConversions(self, conversions, queries):
"""
:type conversions: List[List[int]]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Unit Conversion II
* Difficulty: Medium
* Tags: array, graph, math, 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[][]} conversions
* @param {number[][]} queries
* @return {number[]}
*/
var queryConversions = function(conversions, queries) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Unit Conversion II
* Difficulty: Medium
* Tags: array, graph, math, 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 queryConversions(conversions: number[][], queries: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Unit Conversion II
* Difficulty: Medium
* Tags: array, graph, math, 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[] QueryConversions(int[][] conversions, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Unit Conversion II
```

```
* Difficulty: Medium
* Tags: array, graph, math, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* queryConversions(int** conversions, int conversionsSize, int*
conversionsColSize, int** queries, int queriesSize, int* queriesColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Unit Conversion II
// Difficulty: Medium
// Tags: array, graph, math, 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 queryConversions(conversions [][]int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun queryConversions(conversions: Array<IntArray>, queries: Array<IntArray>):
IntArray {

}
}
```

Swift Solution:

```
class Solution {
func queryConversions(_ conversions: [[Int]], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Unit Conversion II
// Difficulty: Medium
// Tags: array, graph, math, 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 query_conversions(conversions: Vec<Vec<i32>>, queries: Vec<Vec<i32>>)
-> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} conversions
# @param {Integer[][]} queries
# @return {Integer[]}
def query_conversions(conversions, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $conversions
* @param Integer[][] $queries
* @return Integer[]
*/
function queryConversions($conversions, $queries) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<int> queryConversions(List<List<int>> conversions, List<List<int>>
queries) {

}
}
```

Scala Solution:

```
object Solution {
def queryConversions(conversions: Array[Array[Int]], queries:
Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec query_conversions(conversions :: [[integer]], queries :: [[integer]])
:: [integer]
def query_conversions(conversions, queries) do

end
end
```

Erlang Solution:

```
-spec query_conversions(Conversions :: [[integer()]], Queries ::
[[integer()]]) -> [integer()].
query_conversions(Conversions, Queries) ->
.
```

Racket Solution:

```
(define/contract (query-conversions conversions queries)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```