Problem 3528: Unit Conversion I

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

.

Return an array

baseUnitConversion

of length

n

, where

baseUnitConversion[i]

is the number of units of type

i

equivalent to a single unit of type 0. Since the answer may be large, return each

baseUnitConversion[i]

modulo

10

9

+ 7

.

Example 1:

Input:

conversions = [[0,1,2],[1,2,3]]

Output:

[1,2,6]

Explanation:

Convert a single unit of type 0 into 2 units of type 1 using

conversions[0]

.

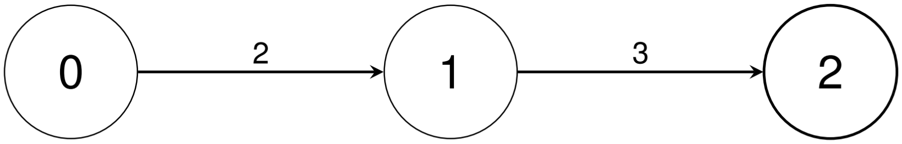
Convert a single unit of type 0 into 6 units of type 2 using

conversions[0]

, then

conversions[1]

.



Example 2:

Input:

conversions = [[0,1,2],[0,2,3],[1,3,4],[1,4,5],[2,5,2],[4,6,3],[5,7,4]]

Output:

[1,2,3,8,10,6,30,24]

Explanation:

Convert a single unit of type 0 into 2 units of type 1 using

conversions[0]

.

Convert a single unit of type 0 into 3 units of type 2 using

conversions[1]

.

Convert a single unit of type 0 into 8 units of type 3 using

conversions[0]

, then

conversions[2]

.

Convert a single unit of type 0 into 10 units of type 4 using

conversions[0]

, then

conversions[3]

.

Convert a single unit of type 0 into 6 units of type 5 using

conversions[1]

, then

conversions[4]

.

Convert a single unit of type 0 into 30 units of type 6 using

conversions[0]

,

conversions[3]

, then

conversions[5]

.

Convert a single unit of type 0 into 24 units of type 7 using

conversions[1]

,

conversions[4]

, then

conversions[6]

.

Constraints:

2 <= n <= 10

5

conversions.length == n - 1

0 <= sourceUnit

i

, targetUnit

i

< n

1 <= conversionFactor

i

<= 10

9

It is guaranteed that unit 0 can be converted into any other unit through a

unique

combination of conversions without using any conversions in the opposite direction.

Code Snippets

C++:

```
class Solution {
public:
vector<int> baseUnitConversions(vector<vector<int>>& conversions) {

}
};
```

Java:

```
class Solution {
public int[] baseUnitConversions(int[][] conversions) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def baseUnitConversions(self, conversions):
```

```
"""
:type conversions: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

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

};
```

TypeScript:

```
function baseUnitConversions(conversions: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] BaseUnitConversions(int[][] conversions) {

}
}
```

C:

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

}
```

Go:

```
func baseUnitConversions(conversions [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun baseUnitConversions(conversions: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func baseUnitConversions(_ conversions: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn base_unit_conversions(conversions: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} conversions
# @return {Integer[]}
def base_unit_conversions(conversions)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $conversions
* @return Integer[]
*/
function baseUnitConversions($conversions) {

}
}
```

Dart:

```
class Solution {
List<int> baseUnitConversions(List<List<int>> conversions) {

}
}
```

Scala:

```
object Solution {
def baseUnitConversions(conversions: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec base_unit_conversions(conversions :: [[integer]]) :: [integer]
def base_unit_conversions(conversions) do

end
end
```

Erlang:

```
-spec base_unit_conversions(Conversions :: [[integer()]]) -> [integer()].
base_unit_conversions(Conversions) ->
.
```

Racket:

```
(define/contract (base-unit-conversions conversions)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Unit Conversion I
* Difficulty: Medium
* 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:
vector<int> baseUnitConversions(vector<vector<int>>& conversions) {

}
};
```

Java Solution:

```
/**
* Problem: Unit Conversion I
* Difficulty: Medium
* 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[] baseUnitConversions(int[][] conversions) {

}
}
```

Python3 Solution:

```
"""
Problem: Unit Conversion I
Difficulty: Medium
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 baseUnitConversions(self, conversions: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Unit Conversion I
* Difficulty: Medium
* 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[][]} conversions
* @return {number[]}
*/
var baseUnitConversions = function(conversions) {
};
```

TypeScript Solution:

```
/**
* Problem: Unit Conversion I
* Difficulty: Medium
* 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 baseUnitConversions(conversions: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Unit Conversion I
* Difficulty: Medium
* 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[] BaseUnitConversions(int[][] conversions) {

}
}
```

C Solution:

```
/*
* Problem: Unit Conversion I
* Difficulty: Medium
* 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
*/

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

}
```

Go Solution:

```
// Problem: Unit Conversion I
// Difficulty: Medium
// 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 baseUnitConversions(conversions [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun baseUnitConversions(conversions: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func baseUnitConversions(_ conversions: [[Int]]) -> [Int] {
}
}
```

Rust Solution:

```
// Problem: Unit Conversion I
// Difficulty: Medium
// 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 base_unit_conversions(conversions: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} conversions
# @return {Integer[]}
def base_unit_conversions(conversions)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $conversions
* @return Integer[]
*/
function baseUnitConversions($conversions) {

}
}
```

Dart Solution:

```
class Solution {
List<int> baseUnitConversions(List<List<int>> conversions) {

}
}
```

Scala Solution:

```
object Solution {
def baseUnitConversions(conversions: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec base_unit_conversions(conversions :: [[integer]]) :: [integer]
def base_unit_conversions(conversions) do

end
end
```

Erlang Solution:

```
-spec base_unit_conversions(Conversions :: [[integer()]]) -> [integer()].
base_unit_conversions(Conversions) ->
.
```

Racket Solution:

```
(define/contract (base-unit-conversions conversions)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```