Problem 2979: Most Expensive Item That Can Not Be Bought

Problem Information

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

Problem Description

You are given two

distinct

prime

numbers

primeOne

and

primeTwo

.

Alice and Bob are visiting a market. The market has an

infinite

number of items, for

any

positive integer

x

there exists an item whose price is

x

. Alice wants to buy some items from the market to gift to Bob. She has an

infinite

number of coins in the denomination

primeOne

and

primeTwo

. She wants to know the

most expensive

item she can

not

buy to gift to Bob.

Return

the price of the

most expensive

item which Alice can not gift to Bob

.

Example 1:

Input:

primeOne = 2, primeTwo = 5

Output:

3

Explanation:

The prices of items which cannot be bought are [1,3]. It can be shown that all items with a price greater than 3 can be bought using a combination of coins of denominations 2 and 5.

Example 2:

Input:

primeOne = 5, primeTwo = 7

Output:

23

Explanation:

The prices of items which cannot be bought are [1,2,3,4,6,8,9,11,13,16,18,23]. It can be shown that all items with a price greater than 23 can be bought.

Constraints:

1 < primeOne, primeTwo < 10

4

primeOne

,

primeTwo

are prime numbers.

primeOne * primeTwo < 10

5

Code Snippets

C++:

```
class Solution {
public:
int mostExpensiveItem(int primeOne, int primeTwo) {

}
};
```

Java:

```
class Solution {
public int mostExpensiveItem(int primeOne, int primeTwo) {

}
}
```

Python3:

```
class Solution:
def mostExpensiveItem(self, primeOne: int, primeTwo: int) -> int:
```

Python:

```
class Solution(object):
def mostExpensiveItem(self, primeOne, primeTwo):
"""
:type primeOne: int
:type primeTwo: int
:rtype: int
```

```
"""
```

JavaScript:

```
/**
* @param {number} primeOne
* @param {number} primeTwo
* @return {number}
*/
var mostExpensiveItem = function(primeOne, primeTwo) {

};
```

TypeScript:

```
function mostExpensiveItem(primeOne: number, primeTwo: number): number {

};
```

C#:

```
public class Solution {
public int MostExpensiveItem(int primeOne, int primeTwo) {

}
}
```

C:

```
int mostExpensiveItem(int primeOne, int primeTwo) {

}
```

Go:

```
func mostExpensiveItem(primeOne int, primeTwo int) int {

}
```

Kotlin:

```
class Solution {
fun mostExpensiveItem(primeOne: Int, primeTwo: Int): Int {

}
}
```

Swift:

```
class Solution {
func mostExpensiveItem(_ primeOne: Int, _ primeTwo: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn most_expensive_item(prime_one: i32, prime_two: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} prime_one
# @param {Integer} prime_two
# @return {Integer}
def most_expensive_item(prime_one, prime_two)

end
```

PHP:

```
class Solution {

/**
* @param Integer $primeOne
* @param Integer $primeTwo
* @return Integer
*/
function mostExpensiveItem($primeOne, $primeTwo) {

}
```

```
}
```

Dart:

```
class Solution {
int mostExpensiveItem(int primeOne, int primeTwo) {

}
}
```

Scala:

```
object Solution {
def mostExpensiveItem(primeOne: Int, primeTwo: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec most_expensive_item(prime_one :: integer, prime_two :: integer) ::
integer
def most_expensive_item(prime_one, prime_two) do

end
end
```

Erlang:

```
-spec most_expensive_item(PrimeOne :: integer(), PrimeTwo :: integer()) ->
integer().
most_expensive_item(PrimeOne, PrimeTwo) ->
.
```

Racket:

```
(define/contract (most-expensive-item primeOne primeTwo)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Most Expensive Item That Can Not Be Bought
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int mostExpensiveItem(int primeOne, int primeTwo) {

}
};
```

Java Solution:

```
/**
* Problem: Most Expensive Item That Can Not Be Bought
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int mostExpensiveItem(int primeOne, int primeTwo) {

}
}
```

Python3 Solution:

```
"""
Problem: Most Expensive Item That Can Not Be Bought
```

```
Difficulty: Medium
Tags: dp, math

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def mostExpensiveItem(self, primeOne: int, primeTwo: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mostExpensiveItem(self, primeOne, primeTwo):
"""
:type primeOne: int
:type primeTwo: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Most Expensive Item That Can Not Be Bought
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} primeOne
* @param {number} primeTwo
* @return {number}
*/
var mostExpensiveItem = function(primeOne, primeTwo) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Most Expensive Item That Can Not Be Bought
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function mostExpensiveItem(primeOne: number, primeTwo: number): number {

};
```

C# Solution:

```
/*
* Problem: Most Expensive Item That Can Not Be Bought
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MostExpensiveItem(int primeOne, int primeTwo) {

}
}
```

C Solution:

```
/*
* Problem: Most Expensive Item That Can Not Be Bought
```

```
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int mostExpensiveItem(int primeOne, int primeTwo) {

}
```

Go Solution:

```
// Problem: Most Expensive Item That Can Not Be Bought
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func mostExpensiveItem(primeOne int, primeTwo int) int {

}
```

Kotlin Solution:

```
class Solution {
fun mostExpensiveItem(primeOne: Int, primeTwo: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func mostExpensiveItem(_ primeOne: Int, _ primeTwo: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Most Expensive Item That Can Not Be Bought
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn most_expensive_item(prime_one: i32, prime_two: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} prime_one
# @param {Integer} prime_two
# @return {Integer}
def most_expensive_item(prime_one, prime_two)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $primeOne
* @param Integer $primeTwo
* @return Integer
*/
function mostExpensiveItem($primeOne, $primeTwo) {

}
}
```

Dart Solution:

```
class Solution {
int mostExpensiveItem(int primeOne, int primeTwo) {

}
}
```

Scala Solution:

```
object Solution {
def mostExpensiveItem(primeOne: Int, primeTwo: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec most_expensive_item(prime_one :: integer, prime_two :: integer) ::
integer
def most_expensive_item(prime_one, prime_two) do

end
end
```

Erlang Solution:

```
-spec most_expensive_item(PrimeOne :: integer(), PrimeTwo :: integer()) ->
integer().
most_expensive_item(PrimeOne, PrimeTwo) ->
.
```

Racket Solution:

```
(define/contract (most-expensive-item primeOne primeTwo)
(-> exact-integer? exact-integer? exact-integer?)
)
```