Problem 991: Broken Calculator

Problem Information

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

Problem Description

There is a broken calculator that has the integer

startValue

on its display initially. In one operation, you can:

multiply the number on display by

2

, or

subtract

1

from the number on display.

Given two integers

startValue

and

target

, return

the minimum number of operations needed to display

target

on the calculator

.

Example 1:

Input:

startValue = 2, target = 3

Output:

2

Explanation:

Use double operation and then decrement operation {2 -> 4 -> 3}.

Example 2:

Input:

startValue = 5, target = 8

Output:

2

Explanation:

Use decrement and then double {5 -> 4 -> 8}.

Example 3:

Input:

startValue = 3, target = 10

Output:

3

Explanation:

Use double, decrement and double {3 -> 6 -> 5 -> 10}.

Constraints:

1 <= startValue, target <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int brokenCalc(int startValue, int target) {

}
};
```

Java:

```
class Solution {
public int brokenCalc(int startValue, int target) {

}
}
```

Python3:

```
class Solution:
def brokenCalc(self, startValue: int, target: int) -> int:
```

Python:

```
class Solution(object):
def brokenCalc(self, startValue, target):
"""
:type startValue: int
:type target: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} startValue
* @param {number} target
* @return {number}
*/
var brokenCalc = function(startValue, target) {

};
```

TypeScript:

```
function brokenCalc(startValue: number, target: number): number {

};
```

C#:

```
public class Solution {
public int BrokenCalc(int startValue, int target) {

}
}
```

C:

```
int brokenCalc(int startValue, int target) {

}
```

Go:

```
func brokenCalc(startValue int, target int) int {

}
```

Kotlin:

```
class Solution {
fun brokenCalc(startValue: Int, target: Int): Int {

}
}
```

Swift:

```
class Solution {
func brokenCalc(_ startValue: Int, _ target: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn broken_calc(start_value: i32, target: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} start_value
# @param {Integer} target
# @return {Integer}
def broken_calc(start_value, target)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer $startValue
* @param Integer $target
* @return Integer
*/
function brokenCalc($startValue, $target) {

}
}
```

Dart:

```
class Solution {
int brokenCalc(int startValue, int target) {

}
}
```

Scala:

```
object Solution {
def brokenCalc(startValue: Int, target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec broken_calc(start_value :: integer, target :: integer) :: integer
def broken_calc(start_value, target) do

end
end
```

Erlang:

```
-spec broken_calc(StartValue :: integer(), Target :: integer()) -> integer().
broken_calc(StartValue, Target) ->
.
```

Racket:

```
(define/contract (broken-calc startValue target)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Broken Calculator
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int brokenCalc(int startValue, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Broken Calculator
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int brokenCalc(int startValue, int target) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Broken Calculator
Difficulty: Medium
Tags: greedy, math

Approach: Greedy algorithm with local optimal choices
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def brokenCalc(self, startValue: int, target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def brokenCalc(self, startValue, target):
"""
:type startValue: int
:type target: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Broken Calculator
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
/**
* @param {number} startValue
* @param {number} target
* @return {number}
*/
var brokenCalc = function(startValue, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Broken Calculator
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function brokenCalc(startValue: number, target: number): number {

};
```

C# Solution:

```
/*
* Problem: Broken Calculator
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int BrokenCalc(int startValue, int target) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Broken Calculator
* Difficulty: Medium
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int brokenCalc(int startValue, int target) {

}
```

Go Solution:

```
// Problem: Broken Calculator
// Difficulty: Medium
// Tags: greedy, math
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func brokenCalc(startValue int, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun brokenCalc(startValue: Int, target: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func brokenCalc(_ startValue: Int, _ target: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Broken Calculator
// Difficulty: Medium
// Tags: greedy, math
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn broken_calc(start_value: i32, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} start_value
# @param {Integer} target
# @return {Integer}
def broken_calc(start_value, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $startValue
* @param Integer $target
* @return Integer
*/
function brokenCalc($startValue, $target) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int brokenCalc(int startValue, int target) {

}
}
```

Scala Solution:

```
object Solution {
def brokenCalc(startValue: Int, target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec broken_calc(start_value :: integer, target :: integer) :: integer
def broken_calc(start_value, target) do

end
end
```

Erlang Solution:

```
-spec broken_calc(StartValue :: integer(), Target :: integer()) -> integer().
broken_calc(StartValue, Target) ->
.
```

Racket Solution:

```
(define/contract (broken-calc startValue target)
(-> exact-integer? exact-integer? exact-integer?)
)
```