Problem 458: Poor Pigs

Problem Information

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

Problem Description

There are

buckets

buckets of liquid, where

exactly one

of the buckets is poisonous. To figure out which one is poisonous, you feed some number of (poor) pigs the liquid to see whether they will die or not. Unfortunately, you only have

minutesToTest

minutes to determine which bucket is poisonous.

You can feed the pigs according to these steps:

Choose some live pigs to feed.

For each pig, choose which buckets to feed it. The pig will consume all the chosen buckets simultaneously and will take no time. Each pig can feed from any number of buckets, and each bucket can be fed from by any number of pigs.

Wait for

minutesToDie

minutes. You may

not

feed any other pigs during this time.

After

minutesToDie

minutes have passed, any pigs that have been fed the poisonous bucket will die, and all others will survive.

Repeat this process until you run out of time.

Given

buckets

,

minutesToDie

, and

minutesToTest

, return

the

minimum

number of pigs needed to figure out which bucket is poisonous within the allotted time

.

Example 1:

Input:

buckets = 4, minutesToDie = 15, minutesToTest = 15

Output:

2

Explanation:

We can determine the poisonous bucket as follows: At time 0, feed the first pig buckets 1 and 2, and feed the second pig buckets 2 and 3. At time 15, there are 4 possible outcomes: - If only the first pig dies, then bucket 1 must be poisonous. - If only the second pig dies, then bucket 3 must be poisonous. - If both pigs die, then bucket 2 must be poisonous. - If neither pig dies, then bucket 4 must be poisonous.

Example 2:

Input:

buckets = 4, minutesToDie = 15, minutesToTest = 30

Output:

2

Explanation:

We can determine the poisonous bucket as follows: At time 0, feed the first pig bucket 1, and feed the second pig bucket 2. At time 15, there are 2 possible outcomes: - If either pig dies, then the poisonous bucket is the one it was fed. - If neither pig dies, then feed the first pig bucket 3, and feed the second pig bucket 4. At time 30, one of the two pigs must die, and the poisonous bucket is the one it was fed.

Constraints:

1 <= buckets <= 1000

1 <= minutesToDie <= minutesToTest <= 100

Code Snippets

C++:

```
class Solution {
public:
int poorPigs(int buckets, int minutesToDie, int minutesToTest) {

}
};
```

Java:

```
class Solution {
public int poorPigs(int buckets, int minutesToDie, int minutesToTest) {

}
}
```

Python3:

```
class Solution:
def poorPigs(self, buckets: int, minutesToDie: int, minutesToTest: int) ->
int:
```

Python:

```
class Solution(object):
def poorPigs(self, buckets, minutesToDie, minutesToTest):
"""
:type buckets: int
:type minutesToDie: int
:type minutesToTest: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} buckets
* @param {number} minutesToDie
* @param {number} minutesToTest
* @return {number}
*/
var poorPigs = function(buckets, minutesToDie, minutesToTest) {

};
```

TypeScript:

```
function poorPigs(buckets: number, minutesToDie: number, minutesToTest:
number): number {

};
```

C#:

```
public class Solution {
public int PoorPigs(int buckets, int minutesToDie, int minutesToTest) {

}
}
```

C:

```
int poorPigs(int buckets, int minutesToDie, int minutesToTest) {

}
```

Go:

```
func poorPigs(buckets int, minutesToDie int, minutesToTest int) int {

}
```

Kotlin:

```
class Solution {
fun poorPigs(buckets: Int, minutesToDie: Int, minutesToTest: Int): Int {

}
}
```

Swift:

```
class Solution {
func poorPigs(_ buckets: Int, _ minutesToDie: Int, _ minutesToTest: Int) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn poor_pigs(buckets: i32, minutes_to_die: i32, minutes_to_test: i32) ->
i32 {

}
}
```

Ruby:

```
# @param {Integer} buckets
# @param {Integer} minutes_to_die
# @param {Integer} minutes_to_test
# @return {Integer}
def poor_pigs(buckets, minutes_to_die, minutes_to_test)

end
```

PHP:

```
class Solution {

/**
* @param Integer $buckets
* @param Integer $minutesToDie
* @param Integer $minutesToTest
* @return Integer
*/
function poorPigs($buckets, $minutesToDie, $minutesToTest) {

}
}
```

Dart:

```
class Solution {
int poorPigs(int buckets, int minutesToDie, int minutesToTest) {

}
}
```

Scala:

```
object Solution {
def poorPigs(buckets: Int, minutesToDie: Int, minutesToTest: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec poor_pigs(buckets :: integer, minutes_to_die :: integer,
minutes_to_test :: integer) :: integer
def poor_pigs(buckets, minutes_to_die, minutes_to_test) do

end
end
```

Erlang:

```
-spec poor_pigs(Buckets :: integer(), MinutesToDie :: integer(),
MinutesToTest :: integer()) -> integer().
poor_pigs(Buckets, MinutesToDie, MinutesToTest) ->
.
```

Racket:

```
(define/contract (poor-pigs buckets minutesToDie minutesToTest)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Poor Pigs
* Difficulty: Hard
* 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 poorPigs(int buckets, int minutesToDie, int minutesToTest) {

}
};
```

Java Solution:

```
/**
* Problem: Poor Pigs
* Difficulty: Hard
* 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 poorPigs(int buckets, int minutesToDie, int minutesToTest) {

}
}
```

Python3 Solution:

```
"""
Problem: Poor Pigs
Difficulty: Hard
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 poorPigs(self, buckets: int, minutesToDie: int, minutesToTest: int) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def poorPigs(self, buckets, minutesToDie, minutesToTest):
"""
:type buckets: int
:type minutesToDie: int
:type minutesToTest: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Poor Pigs
* Difficulty: Hard
* 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} buckets
* @param {number} minutesToDie
* @param {number} minutesToTest
* @return {number}
*/
```

```
var poorPigs = function(buckets, minutesToDie, minutesToTest) {

};
```

TypeScript Solution:

```
/**
* Problem: Poor Pigs
* Difficulty: Hard
* 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 poorPigs(buckets: number, minutesToDie: number, minutesToTest:
number): number {

};
```

C# Solution:

```
/*
* Problem: Poor Pigs
* Difficulty: Hard
* 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 PoorPigs(int buckets, int minutesToDie, int minutesToTest) {

}
}
```

C Solution:

```
/*
* Problem: Poor Pigs
* Difficulty: Hard
* 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 poorPigs(int buckets, int minutesToDie, int minutesToTest) {

}
```

Go Solution:

```
// Problem: Poor Pigs
// Difficulty: Hard
// 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 poorPigs(buckets int, minutesToDie int, minutesToTest int) int {

}
```

Kotlin Solution:

```
class Solution {
fun poorPigs(buckets: Int, minutesToDie: Int, minutesToTest: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func poorPigs(_ buckets: Int, _ minutesToDie: Int, _ minutesToTest: Int) ->
Int {
}
}
```

Rust Solution:

```
// Problem: Poor Pigs
// Difficulty: Hard
// 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 poor_pigs(buckets: i32, minutes_to_die: i32, minutes_to_test: i32) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} buckets
# @param {Integer} minutes_to_die
# @param {Integer} minutes_to_test
# @return {Integer}
def poor_pigs(buckets, minutes_to_die, minutes_to_test)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $buckets
* @param Integer $minutesToDie
* @param Integer $minutesToTest
* @return Integer
*/
function poorPigs($buckets, $minutesToDie, $minutesToTest) {



}
}
```

Dart Solution:

```
class Solution {
int poorPigs(int buckets, int minutesToDie, int minutesToTest) {

}
}
```

Scala Solution:

```
object Solution {
def poorPigs(buckets: Int, minutesToDie: Int, minutesToTest: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec poor_pigs(buckets :: integer, minutes_to_die :: integer,
minutes_to_test :: integer) :: integer
def poor_pigs(buckets, minutes_to_die, minutes_to_test) do

end
end
```

Erlang Solution:

```
-spec poor_pigs(Buckets :: integer(), MinutesToDie :: integer(),
MinutesToTest :: integer()) -> integer().
poor_pigs(Buckets, MinutesToDie, MinutesToTest) ->
.
```

Racket Solution:

```
(define/contract (poor-pigs buckets minutesToDie minutesToTest)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)

)
```