Problem 1689: Partitioning Into Minimum Number Of Deci-Binary Numbers

Problem Information

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

Problem Description

A decimal number is called

deci-binary

if each of its digits is either

0

or

1

without any leading zeros. For example,

101

and

1100

are

deci-binary

, while

112

and

3001

are not.

Given a string

n

that represents a positive decimal integer, return

the

minimum

number of positive

deci-binary

numbers needed so that they sum up to

n

.

Example 1:

Input:

n = "32"

Output:

3

Explanation:

10 + 11 + 11 = 32

Example 2:

Input:

n = "82734"

Output:

8

Example 3:

Input:

n = "27346209830709182346"

Output:

9

Constraints:

1 <= n.length <= 10

5

n

consists of only digits.

n

does not contain any leading zeros and represents a positive integer.

Code Snippets

C++:

```
class Solution {
public:
int minPartitions(string n) {

}
};
```

Java:

```
class Solution {
public int minPartitions(String n) {

}
}
```

Python3:

```
class Solution:
def minPartitions(self, n: str) -> int:
```

Python:

```
class Solution(object):
def minPartitions(self, n):
"""
:type n: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} n
* @return {number}
*/
var minPartitions = function(n) {

};
```

TypeScript:

```
function minPartitions(n: string): number {

};
```

C#:

```
public class Solution {
public int MinPartitions(string n) {

}
}
```

C:

```
int minPartitions(char* n) {

}
```

Go:

```
func minPartitions(n string) int {

}
```

Kotlin:

```
class Solution {
fun minPartitions(n: String): Int {

}
}
```

Swift:

```
class Solution {
func minPartitions(_ n: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_partitions(n: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} n
# @return {Integer}
def min_partitions(n)

end
```

PHP:

```
class Solution {

/**
* @param String $n
* @return Integer
*/
function minPartitions($n) {

}
}
```

Dart:

```
class Solution {
int minPartitions(String n) {

}
}
```

Scala:

```
object Solution {
def minPartitions(n: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_partitions(n :: String.t) :: integer
def min_partitions(n) do

end
end
```

Erlang:

```
-spec min_partitions(N :: unicode:unicode_binary()) -> integer().
min_partitions(N) ->
.
```

Racket:

```
(define/contract (min-partitions n)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Partitioning Into Minimum Number Of Deci-Binary Numbers
* Difficulty: Medium
* Tags: string, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int minPartitions(string n) {

}
};
```

Java Solution:

```
/**
* Problem: Partitioning Into Minimum Number Of Deci-Binary Numbers
* Difficulty: Medium
* Tags: string, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int minPartitions(String n) {

}
}
```

Python3 Solution:

```
"""
Problem: Partitioning Into Minimum Number Of Deci-Binary Numbers
Difficulty: Medium
Tags: string, greedy

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def minPartitions(self, n: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minPartitions(self, n):
"""
:type n: str
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Partitioning Into Minimum Number Of Deci-Binary Numbers
* Difficulty: Medium
* Tags: string, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} n
* @return {number}
*/
var minPartitions = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Partitioning Into Minimum Number Of Deci-Binary Numbers
* Difficulty: Medium
* Tags: string, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function minPartitions(n: string): number {

};
```

C# Solution:

```
/*
* Problem: Partitioning Into Minimum Number Of Deci-Binary Numbers
* Difficulty: Medium
* Tags: string, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MinPartitions(string n) {

}
}
```

C Solution:

```
/*
* Problem: Partitioning Into Minimum Number Of Deci-Binary Numbers
* Difficulty: Medium
* Tags: string, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int minPartitions(char* n) {

}
```

Go Solution:

```
// Problem: Partitioning Into Minimum Number Of Deci-Binary Numbers
// Difficulty: Medium
// Tags: string, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
func minPartitions(n string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minPartitions(n: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func minPartitions(_ n: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Partitioning Into Minimum Number Of Deci-Binary Numbers
// Difficulty: Medium
// Tags: string, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn min_partitions(n: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} n
# @return {Integer}
def min_partitions(n)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $n
* @return Integer
*/
function minPartitions($n) {

}
}
```

Dart Solution:

```
class Solution {
int minPartitions(String n) {

}
}
```

Scala Solution:

```
object Solution {
def minPartitions(n: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_partitions(n :: String.t) :: integer
def min_partitions(n) do

end
end
```

Erlang Solution:

```
-spec min_partitions(N :: unicode:unicode_binary()) -> integer().
min_partitions(N) ->
.
```

Racket Solution:

```
(define/contract (min-partitions n)
(-> string? exact-integer?)
)
```