Problem 433: Minimum Genetic Mutation

Problem Information

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

Problem Description

A gene string can be represented by an 8-character long string, with choices from

'A'

,

'C'

,

'G'

, and

'T'

.

Suppose we need to investigate a mutation from a gene string

startGene

to a gene string

endGene

where one mutation is defined as one single character changed in the gene string.

For example,

"AACCGGTT" --> "AACCGGTA"

is one mutation.

There is also a gene bank

bank

that records all the valid gene mutations. A gene must be in

bank

to make it a valid gene string.

Given the two gene strings

startGene

and

endGene

and the gene bank

bank

, return

the minimum number of mutations needed to mutate from

startGene

to

endGene

. If there is no such a mutation, return

-1

.

Note that the starting point is assumed to be valid, so it might not be included in the bank.

Example 1:

Input:

startGene = "AACCGGTT", endGene = "AACCGGTA", bank = ["AACCGGTA"]

Output:

1

Example 2:

Input:

startGene = "AACCGGTT", endGene = "AAACGGTA", bank = ["AACCGGTA","AACCGCTA","AAACGGTA"]

Output:

2

Constraints:

0 <= bank.length <= 10

startGene.length == endGene.length == bank[i].length == 8

startGene

,

endGene

, and

bank[i]

consist of only the characters

['A', 'C', 'G', 'T']

.

Code Snippets

C++:

```
class Solution {
public:
int minMutation(string startGene, string endGene, vector<string>& bank) {

}
};
```

Java:

```
class Solution {
public int minMutation(String startGene, String endGene, String[] bank) {

}
}
```

Python3:

```
class Solution:
def minMutation(self, startGene: str, endGene: str, bank: List[str]) -> int:
```

Python:

```
class Solution(object):
def minMutation(self, startGene, endGene, bank):
```

```
"""
:type startGene: str
:type endGene: str
:type bank: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} startGene
* @param {string} endGene
* @param {string[]} bank
* @return {number}
*/
var minMutation = function(startGene, endGene, bank) {

};
```

TypeScript:

```
function minMutation(startGene: string, endGene: string, bank: string[]):
number {

};
```

C#:

```
public class Solution {
public int MinMutation(string startGene, string endGene, string[] bank) {

}
}
```

C:

```
int minMutation(char* startGene, char* endGene, char** bank, int bankSize) {

}
```

Go:

```
func minMutation(startGene string, endGene string, bank []string) int {

}
```

Kotlin:

```
class Solution {
fun minMutation(startGene: String, endGene: String, bank: Array<String>): Int
{

}
}
```

Swift:

```
class Solution {
func minMutation(_ startGene: String, _ endGene: String, _ bank: [String]) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_mutation(start_gene: String, end_gene: String, bank: Vec<String>)
-> i32 {

}
}
```

Ruby:

```
# @param {String} start_gene
# @param {String} end_gene
# @param {String[]} bank
# @return {Integer}
def min_mutation(start_gene, end_gene, bank)

end
```

PHP:

```
class Solution {

/**
* @param String $startGene
* @param String $endGene
* @param String[] $bank
* @return Integer
*/
function minMutation($startGene, $endGene, $bank) {

}
}
```

Dart:

```
class Solution {
int minMutation(String startGene, String endGene, List<String> bank) {

}
}
```

Scala:

```
object Solution {
def minMutation(startGene: String, endGene: String, bank: Array[String]): Int
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_mutation(start_gene :: String.t, end_gene :: String.t, bank ::
[String.t]) :: integer
def min_mutation(start_gene, end_gene, bank) do

end
end
```

Erlang:

```
-spec min_mutation(StartGene :: unicode:unicode_binary(), EndGene ::
unicode:unicode_binary(), Bank :: [unicode:unicode_binary()]) -> integer().
min_mutation(StartGene, EndGene, Bank) ->
.
```

Racket:

```
(define/contract (min-mutation startGene endGene bank)
(-> string? string? (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Genetic Mutation
* Difficulty: Medium
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int minMutation(string startGene, string endGene, vector<string>& bank) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Genetic Mutation
* Difficulty: Medium
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

class Solution {
public int minMutation(String startGene, String endGene, String[] bank) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Genetic Mutation
Difficulty: Medium
Tags: string, hash, search

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def minMutation(self, startGene: str, endGene: str, bank: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minMutation(self, startGene, endGene, bank):
"""
:type startGene: str
:type endGene: str
:type bank: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Genetic Mutation
```

```
* Difficulty: Medium
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} startGene
* @param {string} endGene
* @param {string[]} bank
* @return {number}
*/
var minMutation = function(startGene, endGene, bank) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Genetic Mutation
* Difficulty: Medium
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function minMutation(startGene: string, endGene: string, bank: string[]):
number {

};
```

C# Solution:

```
/*
* Problem: Minimum Genetic Mutation
* Difficulty: Medium
* Tags: string, hash, search
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MinMutation(string startGene, string endGene, string[] bank) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Genetic Mutation
* Difficulty: Medium
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minMutation(char* startGene, char* endGene, char** bank, int bankSize) {

}
```

Go Solution:

```
// Problem: Minimum Genetic Mutation
// Difficulty: Medium
// Tags: string, hash, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func minMutation(startGene string, endGene string, bank []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minMutation(startGene: String, endGene: String, bank: Array<String>): Int
{

}
}
```

Swift Solution:

```
class Solution {
func minMutation(_ startGene: String, _ endGene: String, _ bank: [String]) ->
Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Genetic Mutation
// Difficulty: Medium
// Tags: string, hash, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn min_mutation(start_gene: String, end_gene: String, bank: Vec<String>)
-> i32 {

}
}
```

Ruby Solution:

```
# @param {String} start_gene
# @param {String} end_gene
# @param {String[]} bank
# @return {Integer}
def min_mutation(start_gene, end_gene, bank)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $startGene
* @param String $endGene
* @param String[] $bank
* @return Integer
*/
function minMutation($startGene, $endGene, $bank) {

}
}
```

Dart Solution:

```
class Solution {
int minMutation(String startGene, String endGene, List<String> bank) {

}
}
```

Scala Solution:

```
object Solution {
def minMutation(startGene: String, endGene: String, bank: Array[String]): Int
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_mutation(start_gene :: String.t, end_gene :: String.t, bank ::
[String.t]) :: integer
def min_mutation(start_gene, end_gene, bank) do
```

```
end
end
```

Erlang Solution:

```
-spec min_mutation(StartGene :: unicode:unicode_binary(), EndGene ::
unicode:unicode_binary(), Bank :: [unicode:unicode_binary()]) -> integer().
min_mutation(StartGene, EndGene, Bank) ->
.
```

Racket Solution:

```
(define/contract (min-mutation startGene endGene bank)
(-> string? string? (listof string?) exact-integer?)
)
```