

# Problem 187: Repeated DNA Sequences

## Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

The

DNA sequence

is composed of a series of nucleotides abbreviated as

'A'

,

'C'

,

'G'

, and

'T'

.

For example,

"ACGAATTCCG"

is a

DNA sequence

When studying

DNA

, it is useful to identify repeated sequences within the DNA.

Given a string

s

that represents a

DNA sequence

, return all the

10

-letter-long

sequences (substrings) that occur more than once in a DNA molecule. You may return the answer in

any order

Example 1:

Input:

```
s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"
```

Output:

["AAAAACCCCCC", "CCCCCAAAAA"]

Example 2:

Input:

s = "AAAAAAAAAAAAAA"

Output:

["AAAAAAAAAA"]

Constraints:

1 <= s.length <= 10

5

s[i]

is either

'A'

,

'C'

,

'G'

, or

'T'

.

## Code Snippets

### C++:

```
class Solution {  
public:  
vector<string> findRepeatedDnaSequences(string s) {  
  
}  
};
```

### Java:

```
class Solution {  
public List<String> findRepeatedDnaSequences(String s) {  
  
}  
}
```

### Python3:

```
class Solution:  
def findRepeatedDnaSequences(self, s: str) -> List[str]:
```

### Python:

```
class Solution(object):  
def findRepeatedDnaSequences(self, s):  
    """  
    :type s: str  
    :rtype: List[str]  
    """
```

### JavaScript:

```
/**  
 * @param {string} s  
 * @return {string[]}  
 */  
var findRepeatedDnaSequences = function(s) {
```

```
};
```

### TypeScript:

```
function findRepeatedDnaSequences(s: string): string[] {  
}  
};
```

### C#:

```
public class Solution {  
    public IList<string> FindRepeatedDnaSequences(string s) {  
        }  
    }  
}
```

### C:

```
/**  
 * Note: The returned array must be malloced, assume caller calls free().  
 */  
char** findRepeatedDnaSequences(char* s, int* returnSize) {  
}
```

### Go:

```
func findRepeatedDnaSequences(s string) []string {  
}  
}
```

### Kotlin:

```
class Solution {  
    fun findRepeatedDnaSequences(s: String): List<String> {  
        }  
    }  
}
```

### Swift:

```
class Solution {  
    func findRepeatedDnaSequences(_ s: String) -> [String] {  
        }  
    }  
}
```

**Rust:**

```
impl Solution {  
    pub fn find_repeated_dna_sequences(s: String) -> Vec<String> {  
        }  
    }  
}
```

**Ruby:**

```
# @param {String} s  
# @return {String[]}  
def find_repeated_dna_sequences(s)  
  
end
```

**PHP:**

```
class Solution {  
  
    /**  
     * @param String $s  
     * @return String[]  
     */  
    function findRepeatedDnaSequences($s) {  
  
    }  
}
```

**Dart:**

```
class Solution {  
    List<String> findRepeatedDnaSequences(String s) {  
        }  
    }
```

### Scala:

```
object Solution {  
    def findRepeatedDnaSequences(s: String): List[String] = {  
  
    }  
}
```

### Elixir:

```
defmodule Solution do  
  @spec find_repeated_dna_sequences(s :: String.t) :: [String.t]  
  def find_repeated_dna_sequences(s) do  
  
  end  
end
```

### Erlang:

```
-spec find_repeated_dna_sequences(S :: unicode:unicode_binary()) ->  
[unicode:unicode_binary()].  
find_repeated_dna_sequences(S) ->  
.
```

### Racket:

```
(define/contract (find-repeated-dna-sequences s)  
  (-> string? (listof string?))  
)
```

## Solutions

### C++ Solution:

```
/*  
 * Problem: Repeated DNA Sequences  
 * Difficulty: Medium  
 * Tags: array, string, tree, hash  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(h) for recursion stack where h is height
```

```

*/
class Solution {
public:
vector<string> findRepeatedDnaSequences(string s) {

}
};


```

### Java Solution:

```

/**
 * Problem: Repeated DNA Sequences
 * Difficulty: Medium
 * Tags: array, string, tree, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
public List<String> findRepeatedDnaSequences(String s) {

}
}


```

### Python3 Solution:

```

"""
Problem: Repeated DNA Sequences
Difficulty: Medium
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def findRepeatedDnaSequences(self, s: str) -> List[str]:

```

```
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
    def findRepeatedDnaSequences(self, s):
        """
        :type s: str
        :rtype: List[str]
        """

```

### JavaScript Solution:

```
/**
 * Problem: Repeated DNA Sequences
 * Difficulty: Medium
 * Tags: array, string, tree, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * @param {string} s
 * @return {string[]}
 */
var findRepeatedDnaSequences = function(s) {

};
```

### TypeScript Solution:

```
/**
 * Problem: Repeated DNA Sequences
 * Difficulty: Medium
 * Tags: array, string, tree, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
```

```

* Space Complexity: O(h) for recursion stack where h is height
*/



function findRepeatedDnaSequences(s: string): string[] {
}

```

### C# Solution:

```

/*
 * Problem: Repeated DNA Sequences
 * Difficulty: Medium
 * Tags: array, string, tree, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

public class Solution {
    public IList<string> FindRepeatedDnaSequences(string s) {
        return null;
    }
}

```

### C Solution:

```

/*
 * Problem: Repeated DNA Sequences
 * Difficulty: Medium
 * Tags: array, string, tree, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

/***
 * Note: The returned array must be malloced, assume caller calls free().
 */
char** findRepeatedDnaSequences(char* s, int* returnSize) {

```

```
}
```

### Go Solution:

```
// Problem: Repeated DNA Sequences
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func findRepeatedDnaSequences(s string) []string {
}
```

### Kotlin Solution:

```
class Solution {
    fun findRepeatedDnaSequences(s: String): List<String> {
        return emptyList()
    }
}
```

### Swift Solution:

```
class Solution {
    func findRepeatedDnaSequences(_ s: String) -> [String] {
        return []
    }
}
```

### Rust Solution:

```
// Problem: Repeated DNA Sequences
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
    pub fn find_repeated_dna_sequences(s: String) -> Vec<String> {
        }

    }
}
```

### Ruby Solution:

```
# @param {String} s
# @return {String[]}
def find_repeated_dna_sequences(s)

end
```

### PHP Solution:

```
class Solution {

    /**
     * @param String $s
     * @return String[]
     */
    function findRepeatedDnaSequences($s) {

    }
}
```

### Dart Solution:

```
class Solution {
    List<String> findRepeatedDnaSequences(String s) {
        }

    }
}
```

### Scala Solution:

```
object Solution {
    def findRepeatedDnaSequences(s: String): List[String] = {
```

```
}
```

```
}
```

### Elixir Solution:

```
defmodule Solution do
  @spec find_repeated_dna_sequences(s :: String.t) :: [String.t]
  def find_repeated_dna_sequences(s) do
    end
  end
```

### Erlang Solution:

```
-spec find_repeated_dna_sequences(S :: unicode:unicode_binary()) ->
[unicode:unicode_binary()].
find_repeated_dna_sequences(S) ->
.
```

### Racket Solution:

```
(define/contract (find-repeated-dna-sequences s)
(-> string? (listof string?))
)
```